US007441090B2

(12) **United States Patent**
Estakhri et al.

(10) **Patent No.: US 7,441,090 B2**
(45) **Date of Patent: *Oct. 21, 2008**

(54) **SYSTEM AND METHOD FOR UPDATING DATA SECTORS IN A NON-VOLATILE MEMORY USING LOGICAL BLOCK ADDRESSING**

(75) Inventors: **Petro Estakhri**, Pleasanton, CA (US); **Berhanu Iman**, Sunnyvale, CA (US); **Ali Ganjuei**, deceased, late of San Ramon CA (US); by **Joumana Fahim**, legal representative, Dubai (AE)

(73) Assignee: **Lexar Media, Inc.**, Fremont, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 218 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/201,612**

(22) Filed: **Aug. 10, 2005**

(65) **Prior Publication Data**

US 2006/0020747 A1 Jan. 26, 2006

Related U.S. Application Data

(63) Continuation of application No. 09/620,544, filed on Jul. 21, 2000, now Pat. No. 6,978,342, which is a continuation of application No. 09/264,340, filed on Mar. 8, 1999, now Pat. No. 6,145,051, which is a continuation of application No. 08/831,266, filed on Mar. 31, 1997, now Pat. No. 5,907,856, which is a continuation-in-part of application No. 08/509,706, filed on Jul. 31, 1995, now Pat. No. 5,845,313.

(51) **Int. Cl.**
G06F 12/12 (2006.01)
G06F 12/06 (2006.01)

(52) **U.S. Cl.** **711/159**; 711/203; 711/209

(58) **Field of Classification Search** ..... None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,099,069 A 7/1978 Cricchi et al.

(Continued)

FOREIGN PATENT DOCUMENTS

AU 0 557 723 1/1987

(Continued)

OTHER PUBLICATIONS

Book—*Computer Architecture and Parallel Processing*, Kai Hwang & Faye A. Briggs, McGraw-Hill Book Co., © 1984, p. 64.

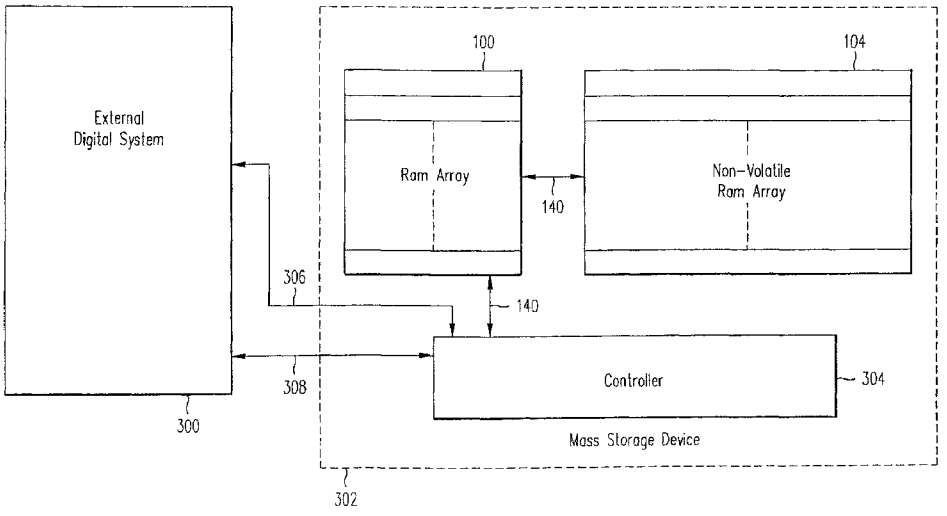
(Continued)

Primary Examiner—B. James Peikari
(74) *Attorney, Agent, or Firm*—Leffert, Jay & Polglaze, P.A.

(57) **ABSTRACT**

A device is disclosed for storing mapping information for mapping a logical block address identifying a block being accessed by a host to a physical block address, identifying a free area of nonvolatile memory, the block being selectively erasable and having one or more sectors that may be individually moved. The mapping information including a virtual physical block address for identifying an "original" location, within the nonvolatile memory, wherein a block is stored and a moved virtual physical block address for identifying a "moved" location, within the nonvolatile memory, wherein one or more sectors of the stored block are moved. The mapping information further including status information for use of the "original" physical block address and the "moved" physical block address and for providing information regarding "moved" sectors within the block being accessed.

11 Claims, 19 Drawing Sheets



U.S. PATENT DOCUMENTS

4,130,900	A		12/1978	Watanabe
4,210,959	A		7/1980	Wozniak
4,309,627	A		1/1982	Tabata
4,355,376	A		10/1982	Gould
4,398,248	A		8/1983	Hsia et al.
4,405,952	A		9/1983	Slakmon
4,414,627	A		11/1983	Nakamura
4,450,559	A		5/1984	Bond et al.
4,456,971	A		6/1984	Fukuda et al.
4,468,730	A		8/1984	Dodd et al.
4,473,878	A		9/1984	Zolnowsky et al.
4,476,526	A		10/1984	Dodd
4,498,146	A		2/1985	Martinez
4,525,839	A		6/1985	Nozawa et al.
4,532,590	A		7/1985	Wallach et al.
4,609,833	A		9/1986	Gutterman
4,616,311	A		10/1986	Sato
4,654,847	A		3/1987	Dutton
4,710,871	A		12/1987	Belknap et al.
4,746,998	A		5/1988	Robinson et al.
4,748,320	A		5/1988	Yorimoto et al.
4,757,474	A		7/1988	Fukushi et al.
4,774,700	A		9/1988	Satoh et al.
4,780,855	A		10/1988	Iida et al.
4,788,665	A		11/1988	Fukuda et al.
4,797,543	A		1/1989	Watanabe
4,800,520	A		1/1989	Iijima
4,829,169	A		5/1989	Watanabe
4,843,224	A		6/1989	Ohta et al.
4,896,262	A		1/1990	Wayama et al.
4,914,529	A		4/1990	Bonke
4,920,518	A		4/1990	Nakamura et al.
4,924,331	A		5/1990	Robinson et al.
4,943,745	A		7/1990	Watanabe et al.
4,953,122	A		8/1990	Williams
4,970,642	A		11/1990	Yamamura
4,970,727	A		11/1990	Miyawaki et al.
5,070,474	A		12/1991	Tuma et al.
5,093,785	A		3/1992	Iljima
5,168,465	A		12/1992	Harari
5,198,380	A		3/1993	Harari
5,200,959	A		4/1993	Gross et al.
5,218,695	A		6/1993	Noveck et al.
5,220,518	A		6/1993	Haq
5,226,168	A		7/1993	Kobayashi et al.
5,227,714	A		7/1993	Lou
5,253,351	A		10/1993	Yamamoto et al.
5,267,218	A		11/1993	Elbert
5,268,318	A		12/1993	Harari
5,268,870	A		12/1993	Harari
5,270,979	A		12/1993	Harari et al.
5,293,560	A		3/1994	Harari
5,297,148	A		3/1994	Harari et al.
5,303,198	A		4/1994	Adachi et al.
5,305,276	A		4/1994	Uenoyama
5,305,278	A		4/1994	Inoue
5,315,541	A		5/1994	Harari et al.
5,315,558	A		5/1994	Hag
5,329,491	A		7/1994	Brown et al.
5,337,275	A		8/1994	Garner
5,341,330	A		8/1994	Wells et al.
5,341,339	A		8/1994	Wells
5,341,341	A		8/1994	Fukazo
5,353,256	A		10/1994	Fandrich et al.
5,357,475	A		10/1994	Hasbun et al.
5,359,569	A		10/1994	Fujita et al.
5,365,127	A		11/1994	Manley
5,369,615	A		11/1994	Harari et al.
5,371,702	A		12/1994	Nakai et al.
5,381,539	A		1/1995	Yanai et al.
5,382,839	A		1/1995	Shinohara
5,384,743	A		1/1995	Rouy
5,388,083	A		2/1995	Assar et al.
5,396,468	A		3/1995	Harari et al.
5,404,485	A		4/1995	Ban
5,406,527	A		4/1995	Honma
5,418,752	A		5/1995	Harari et al.
5,422,842	A		6/1995	Cernea et al.
5,422,856	A		6/1995	Sasaki et al.
5,428,621	A		6/1995	Mehrotra et al.
5,430,682	A		7/1995	Ishikawa et al.
5,430,859	A		7/1995	Norman et al.
5,431,330	A		7/1995	Wieres
5,434,825	A		7/1995	Harari
5,438,573	A		8/1995	Mangan et al.
5,455,926	A	*	10/1995	Keele et al. 711/4
5,465,235	A		11/1995	Miyamoto
5,465,338	A		11/1995	Clay
5,471,478	A		11/1995	Mangan et al.
5,473,765	A		12/1995	Gibbons et al.
5,479,638	A		12/1995	Assar et al.
5,485,595	A		1/1996	Assar et al.
5,490,117	A		2/1996	Oda et al.
5,495,442	A		2/1996	Cernea et al.
5,504,760	A		4/1996	Harari et al.
5,508,971	A		4/1996	Cernea et al.
5,513,138	A		4/1996	Manabe et al.
5,515,333	A		5/1996	Fujita et al.
5,519,847	A		5/1996	Fandrich et al.
5,523,980	A		6/1996	Sakui et al.
5,524,230	A		6/1996	Sakaue et al.
5,530,673	A		6/1996	Tobita et al.
5,530,828	A		6/1996	Kaki et al.
5,530,938	A		6/1996	Akasaka et al.
5,532,962	A		7/1996	Auclair et al.
5,532,964	A		7/1996	Cernea et al.
5,534,456	A		7/1996	Yuan et al.
5,535,328	A		7/1996	Harari et al.
5,541,551	A		7/1996	Brehner et al.
5,544,118	A		8/1996	Harari
5,544,356	A		8/1996	Robinson et al.
5,552,698	A		9/1996	Tai et al.
5,554,553	A		9/1996	Harari
5,563,825	A		10/1996	Cernea et al.
5,566,314	A		10/1996	DeMarco et al.
5,568,439	A		10/1996	Harari
5,572,466	A		11/1996	Sukegawa
5,579,502	A		11/1996	Konishi et al.
5,581,723	A		12/1996	Hasbun et al.
5,583,812	A		12/1996	Harari
5,592,415	A		1/1997	Kato et al.
5,592,420	A		1/1997	Cernea et al.
5,596,526	A		1/1997	Assar et al.
5,598,370	A		1/1997	Niisima et al.
5,602,987	A		2/1997	Harari et al.
5,603,001	A		2/1997	Sukegawa et al.
5,606,660	A		2/1997	Estakhri et al.
5,611,067	A		3/1997	Okamoto et al.
5,640,528	A		6/1997	Harney et al.
5,642,312	A		6/1997	Harari
5,648,929	A		7/1997	Miyamoto
5,663,901	A		9/1997	Wallace et al.
5,693,570	A		12/1997	Cernea et al.
5,712,819	A		1/1998	Harari
5,719,808	A		2/1998	Harari et al.
5,723,990	A		3/1998	Roohparvar
5,734,567	A		3/1998	Griffiths et al.
5,734,816	A		3/1998	NiiJima et al.
5,740,396	A		4/1998	Mason
5,745,418	A		4/1998	Ma et al.
5,754,567	A		5/1998	Norman
5,757,712	A		5/1998	Nagel et al.
5,758,100	A		5/1998	Odisho
5,761,117	A		6/1998	Uchino et al.

5,768,190	A	6/1998	Tanaka et al.
5,768,195	A	6/1998	Nakamura et al.
5,773,901	A	6/1998	Kanter
5,778,418	A	7/1998	Auclair et al.
5,781,478	A	7/1998	Takeuchi et al.
5,787,445	A	7/1998	Daberko
5,787,484	A	7/1998	Norman
RE35,881	E	8/1998	Barrett et al.
5,799,168	A	8/1998	Ban
5,802,551	A	9/1998	Komatsu et al.
5,809,515	A	9/1998	Kaki et al.
5,809,558	A	9/1998	Matthews et al.
5,809,560	A	9/1998	Schneider
5,818,350	A	10/1998	Estakhri et al.
5,818,781	A	10/1998	Estakhri et al.
5,822,245	A	10/1998	Gupta et al.
5,822,252	A	10/1998	Lee et al.
5,822,781	A	10/1998	Wells et al.
5,831,929	A	11/1998	Manning
5,835,935	A	11/1998	Estakhri et al.
5,838,614	A	11/1998	Estakhri et al.
5,845,313	A	12/1998	Estakhri et al.
5,847,552	A	12/1998	Brown
5,860,083	A	1/1999	Sukeawa
5,860,124	A	1/1999	Matthews et al.
5,862,099	A	1/1999	Gannage et al.
5,890,192	A	3/1999	Lee et al.
5,901,086	A	5/1999	Wang et al.
5,907,856	A	5/1999	Estakhri et al.
5,909,586	A	6/1999	Anderson
5,920,884	A	7/1999	Jennings, III et al.
5,924,113	A	7/1999	Estakhri et al.
5,928,370	A	7/1999	Asnaashari
5,930,815	A	7/1999	Estakhri et al.
5,933,368	A	8/1999	Ma et al.
5,933,846	A	8/1999	Endo
5,936,971	A	8/1999	Harari et al.
5,937,425	A	8/1999	Ban
5,953,737	A	9/1999	Estakhri et al.
5,956,473	A	9/1999	Ma et al.
5,959,926	A	9/1999	Jones et al.
5,966,727	A	10/1999	Nishino et al.
5,986,933	A	11/1999	Takeuchi et al.
5,987,563	A	11/1999	Itoh et al.
5,987,573	A	11/1999	Hiraka
5,991,849	A	11/1999	Yamada et al.
6,011,322	A	1/2000	Stumfall et al.
6,011,323	A	1/2000	Camp
6,018,265	A	1/2000	Keshtbod
6,021,408	A	2/2000	Ledain et al.
6,026,020	A	2/2000	Matsubara et al.
6,026,027	A	2/2000	Terrell, II et al.
6,034,897	A	3/2000	Estakhri et al.
6,035,357	A	3/2000	Sakaki
6,040,997	A	3/2000	Estakrhi
6,041,001	A	3/2000	Estakhri
6,047,352	A	4/2000	Lakhani et al.
6,055,184	A	4/2000	Acharya et al.
6,055,188	A	4/2000	Takeuchi et al.
6,069,827	A	5/2000	Sinclair
6,072,796	A	6/2000	Christensen et al.
6,076,137	A	6/2000	Asnaashari
6,081,447	A	6/2000	Lofgren et al.
6,081,878	A	6/2000	Estakhri et al.
6,084,483	A	7/2000	Keshtbod
6,097,666	A	8/2000	Sakui et al.
6,115,785	A	9/2000	Estakhri et al.
6,122,195	A	9/2000	Estakhri et al.
6,125,424	A	9/2000	Komatsu et al.
6,125,435	A	9/2000	Estakhri et al.
6,128,695	A	10/2000	Estakhri et al.
6,134,145	A	10/2000	Wong
6,134,151	A	10/2000	Estakhri et al.
6,141,249	A	10/2000	Estakhri et al.
6,145,051	A	11/2000	Estakhri et al.
6,151,247	A	11/2000	Estakhri et al.
6,172,906	B1	1/2001	Estakhri et al.
6,173,362	B1	1/2001	Yoda
6,181,118	B1	1/2001	Meehan et al.
6,182,162	B1	1/2001	Estakhri et al.
6,202,138	B1	3/2001	Estakhri et al.
6,223,308	B1	4/2001	Estakhri et al.
6,226,708	B1	5/2001	McGoldrick et al.
6,230,234	B1	5/2001	Estakhri et al.
6,262,918	B1	7/2001	Estakhri et al.
6,272,610	B1	8/2001	Katayama et al.
6,275,436	B1	8/2001	Tobita et al.
6,279,069	B1	8/2001	Robinson et al.
6,279,114	B1	8/2001	Toombs et al.
6,285,607	B1	9/2001	Sinclair
6,327,639	B1	12/2001	Asnaashari
6,345,367	B1	2/2002	Sinclair
6,374,337	B1	4/2002	Estakhri
6,385,667	B1	5/2002	Estakhri et al.
6,393,513	B2	5/2002	Estakhri et al.
6,397,314	B1	5/2002	Estakhri et al.
6,411,546	B1	6/2002	Estakhri et al.
6,467,021	B1	10/2002	Sinclair
6,490,649	B2	12/2002	Sinclair
6,567,307	B1	5/2003	Estakhri
6,578,127	B1	6/2003	Sinclair
6,587,382	B1	7/2003	Estakhri et al.
6,711,059	B2	3/2004	Sinclair et al.
6,721,819	B2	4/2004	Estakhri et al.
6,721,843	B1	4/2004	Estakhri
6,725,321	B1	4/2004	Sinclair et al.
6,728,851	B1	4/2004	Estakhri et al.
6,751,155	B2	6/2004	Gorobets
6,757,800	B1	6/2004	Estakhri et al.
6,772,274	B1	8/2004	Estakhri
6,813,678	B1	11/2004	Sinclair et al.
6,898,662	B2	5/2005	Gorobets
6,912,618	B2	6/2005	Estakhri et al.
6,950,918	B1	9/2005	Estakhri
6,957,295	B1	10/2005	Estakhri
6,973,519	B1	12/2005	Estakhri et al.
6,978,342	B1	12/2005	Estakhri et al.
7,000,064	B2	2/2006	Payne et al.
2003/0033471	A1	2/2003	Lin et al.

FOREIGN PATENT DOCUMENTS

EP	0 220 718	A2	5/1987
EP	0 243 503	A1	11/1987
EP	0 392 895	A2	10/1990
EP	0 424 191	A2	4/1991
EP	0 489 204	A1	6/1992
EP	0 522 780	A2	1/1993
EP	0 522 780	B1	1/1993
EP	0 544 252	A2	6/1993
EP	0 613 151	A2	8/1994
EP	0 617 363	A2	9/1994
EP	0 619 541	A2	10/1994
EP	0 663 636	A1	7/1995
EP	0 686 976	A2	12/1995
EP	0 897 579	B1	7/2000
EP	0 891 580	B1	11/2000
EP	0 896 699	B1	11/2000
EP	0 852 766	B1	5/2001
EP	0 852 765	B1	9/2001
EP	0 722 585	B1	5/2002
EP	0 910 826	B1	6/2002
EP	0 691 008	B1	11/2002
EP	0 861 468	B1	4/2003
EP	0 978 040	B1	5/2004
EP	1 157 328	B1	5/2005

FR	93 01908		8/1993
GB	2 251 323	A	7/1992
GB	2 291 990	A	2/1996
GB	2 304 428	A	3/1997
GB	2 348 991	B	12/2002
GB	2 351 822	B	1/2003
GB	2 384 337	A	7/2003
GB	2 384 883	A	10/2005
GB	2 384 338	B	11/2005
GB	2 384 072	B	12/2005
GB	2 411 499	B	2/2006
IS	117881		5/2003
JP	59-45695	A	7/1982
JP	58-215794	A	12/1983
JP	58-215795	A	12/1983
JP	59-092483		5/1984
JP	59-162695	A	9/1984
JP	60-212900		10/1985
JP	61-96598	A	5/1986
JP	62-283496	A	12/1987
JP	62-283497	A	12/1987
JP	63-183700	A	7/1988
JP	1-138694		5/1989
JP	3-228377		10/1991
JP	4-057295		2/1992
JP	4-254994		9/1992
JP	4-268284		9/1992
JP	4-278297		10/1992
JP	4-332999	A	11/1992
JP	5-128877		5/1993
JP	5-282883		10/1993
JP	6-004399		1/1994
JP	6-036578		2/1994
JP	6-124175		5/1994
JP	6-124231		5/1994
JP	6-131889		5/1994
JP	6-132747		5/1994
JP	6-149395		5/1994
JP	6-266596		9/1994
JP	7-084871		3/1995
JP	7-093499		4/1995
JP	7-114499		5/1995
JP	7-141258		6/1995
JP	7-235193		9/1995
JP	7-311708		11/1995
JP	7-334996		12/1995
JP	8-018018		1/1996
JP	8-069696		3/1996
JP	9-147581		6/1997
SU	1388877	A1	4/1988
SU	1408439	A1	7/1988
SU	1515164	A1	10/1989
SU	1541619	A1	2/1990
SU	1573458	A2	6/1990
SU	1686449	A2	10/1991
WO	84/00628		2/1984
WO	WO 94/20906		9/1994

OTHER PUBLICATIONS

Magazine—"State of the Art: Magnetic VS. Optical Store Data in a Flash", by Walter Lahti and Dean McCarron, Byte Magazine dated Nov. 1, 1990. 311, vol. 15, No. 12.

Magazine—Technology Updates, Intergrated Cirrcuits, "1-Mbit flash memories seek their role in system design", Ron Wilson, Senior Editor, Computer Design magazine 28 (1989) Mar. 1, No. 5, Tulsa OK, US, pp. 30 and 32.

1992 Symposium of VLSI Circuits Digest of Technical Papers, "EEPROM for Solid State Disk Applications", S. Mehoura et al., SunDisk Corporation, Santa Clara, CA. R.W. Gregor et al., AT&T Bell Laboratories, Allentown, PA. pp. 24 and 25.

Mendel Rosenblum and John K. Ousterhout, The Design and Implementation of a Log-Structured File System, article, 1991, 15 pgs., Berkeley, USA.

Brian Dipert and Markus Levy, Designing with Flash Memory, book, Apr. 1994, 445 pgs., Annabooks, San Diego, USA.

Science Forum, Inc., Flash Memory Symposium '95,symposium, 1995, 13 pgs.; Hongo,Bunkyo-ku, Tokyo.

Ross S. Finlayson and David R. Cheriton, An Extended File Service Exploiting Write-Once Storage, article, 1987, 10 pgs. ACM.

Jason Gait, The Optical File Cabinet: A Random-Access File System for Write-Once Optical Disks, article, Jun. 1988, 12 pgs., Beaverton, Oregon.

Henry G. Baker, Memory Management, book, 1995, 19 pgs., Springer-Verlag Berlin Heidelberg, Germany.

Sape J. Mullender and Andrew S. Tanenbaum, A Distributed File Service Based on Optimistic Concurrency Control, article, 1985, 12 pgs., ACM.

Hiroshi Nakamura, Junichi Miyamoto, Kenichi Imamiya and Yoshihisa Iwata, A Novel Sense Amplifier for Flexible Voltage Operation NAND Flash Memories, symposium, 1995, VLSI Circuits Digest of Technical Papers,, 2 pgs.

Hiroshi Nakamura, Junichi Miyamoto, Kenichi Imamiya, Yoshihisa Iwata, Yoshihisa Sugiura and Hideko Oodaira, A Novel Sensing Scheme with On-Chip Page Copy for Flexible Voltage NAND Flash Memories, article, Jun. 1996, 9 pgs . . . vol. E79-C, No. 6.

Takaaki Nozaki, Toshiaki Tanaka, Yoshiro Kijiya, Eita Kinoshita, Tatsuo Tsuchiya and Yutaka Hayashi, A 1-Mb EEPROM with MONOS Memory Cell for Semiconductor Disk Application, article, 1991, 5 pgs., Journal Of Solid -State Circuits, vol. 26, No. 4.

Kai Hwang and Faye A. Briggs. Computer Architecture and Parallel Processing, book, 1984, McGraw-Hill, Inc., 2 pgs., US.

Walter Lahti and Dean McCarron, State of the Art: Magnetic VS. Optical Store Data in a Flash, article, 1990, 7 pgs., vol. 15, No. 12, McGraw-Hill, Inc., US.

Ron Wilson, Integrated Circuits: 1-Mbit flash memories seek their role in system design, article, Mar. 1, 1989. 2 pgs., No. 6, Tulsa, OK.

S. Mehroura, J.H. Yuan, R.A Cemea, W.Y. Chien, D.C. Guteman, G. Samachisa, R.D. Noman, M. Mofidi, W.Lee. Y. Fong, A. Mihnea, E. Hann, R.W. Gregor, E.P. Eberhardt, J.R. Radosevich, K.R. Stiles, R.A. Kohler, C.W. Leung, and T.J. Mulrooney, Serial 9Mb F EEPROM for Solid State Disk Applications, symposium, 1992, 2 pgs., Mountain View, CA.

Steven H. Leibson, Nonvolatile, in-circuit-reprogrammable memories, article, Jan. 3, 1991. 12 pgs., EDN, Circle No. 12.

Walter Lahti and Dean McCarron, State of the Art: Magnetic VS. Optical Store Data in a Flash, article. 1990, 7 pgs., vol. 15, No. 12, McGraw-Hill, Inc., US.

Ramon Caceres, Fred Douglis, Kai Li and Brian Marsh, Operationg System Implications of Solid-State Mobile Computers, article, 7 pgs., Oct. 1993, Workshop on Workstation Operting Systems.

Michael Wu and Wily Zwaenepoel, A Non-Volatile, Main Memory Storage System, 12 pgs., 1994, ACM, San Jose, CA USA.

Dave Bursky, Innovative flash memories match DRAM densities: available with a choice of features, flash memories are finding homes in many systems (includes related articles on the origins of flash, and on the differences between NAND and NOR flash memories), article, May 16, 1994, 9 pgs., Electronic Design, v.42, N. 10, The Gale Group.

Anthony Cataldo, New flash enhancements up ante. (Intel's 28F400BV-120 and 28F004BV-120, Atmel's AT29BV010 and AT29BV020, and Samsung Semiconductor's KM29V32000 "flash" memory* device)(Product Announcement). article, Mar. 13, 1995, 4 pgs., Electronic News. v.41, n.2056, The Gale Group.

Sam Weber, "Flash" modules' portability, reusability, small size valued for a host of APPs-Consumer formals flocking to "flash", article, Jul. 22, 1996, 9 pgs. Electronic Engieering Times, n.911, CMP Media.

Toshiba, MOS Memory (Non-Volatile), 1995, Data Book.

Stan Baker, But Integration Calls for Hardware, Software Changes: Flash: designers face the dawn of a new memory age, article, Sep. 12, 2003, 5 pgs., Electronic Engineering Times. 1990. N.619. 41, CMP Media.

Toshiba, Toshiba MOS Digital Integrated circuit silicon gate CMOS, (TC58NS512OC) Mar. 21, 2001, 43 pgs., Data Book.

Toshiba Corporation, SMIL (Smartmedia Interface Library) Hardware Edition Version 1.00, Jul. 1, 2000, 36 pgs., Data Book.

Toshiba, Toshiba MOS Digital Integrated Circuit Silicon Gate, (TC58512FT), Mar. 05, 2001, 43 pgs., Data Book.

Toshiba, Toshiba MOS Digital Integrated Circuit Silicon Gate, (TC58DVM92A1FT00), Jan. 10, 2003, 44 pgs., Data Book.

Toshiba, Toshiba MOS Digital Integrated Circuit Silicon Gate, (TC58DVG02A1FT00), Jan. 10, 2003, 44 pgs., Data Book.

Toshiba, Toshiba MOS Digital Integrated Circuit Silicon Gate, (TH58100FT), Mar. 5, 2001, 43 pgs., Data Book.

35Nonvolatile Memory Technology Review, A Time of Change, Proceedings 1993 Conference, Jun. 22-24, 1993, Linthlcum Heights, MD USA.

Toshiba Corporation, SMIL (Smartmedia Interface Library) Software Edition Version 1.00, Jul. 1, 2000, 136 pgs., Data Book.

Toshiba, MOS Memory (Non-Volatile), 1996, 279 pgs., Data Book.

Dan Auclair, Optimal Solid State Disk Architecure For Portable Computers, symposium, Jul. 9, 1991, 7 pgs., SunDisk Corporation.

* cited by examiner

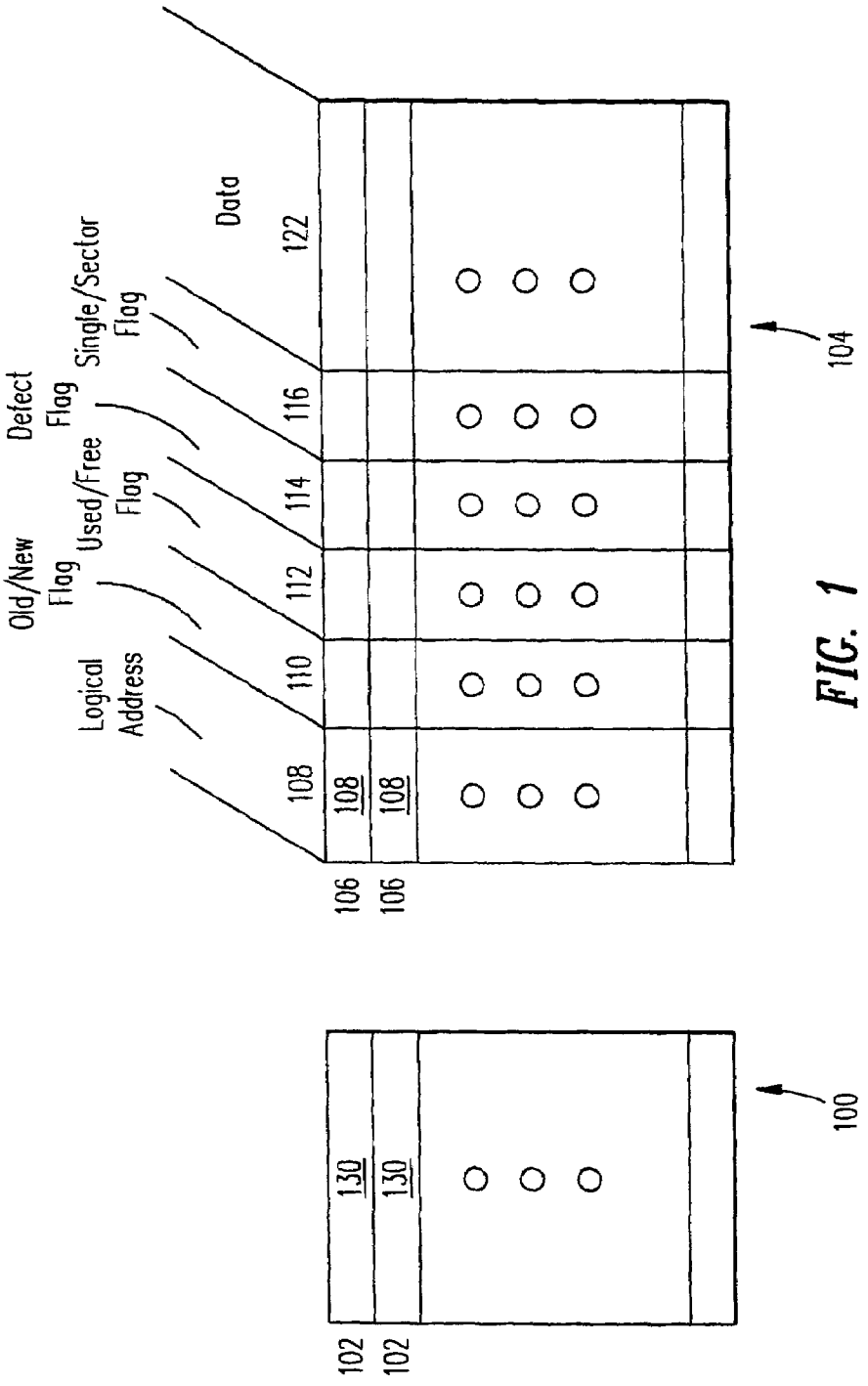

Logical Address
Old/New Flag
Used/Free Flag
Defect Flag
Single/Sector Flag
Data
108
110
112
114
116
122
106
106
104
102
102
100

FIG. 1

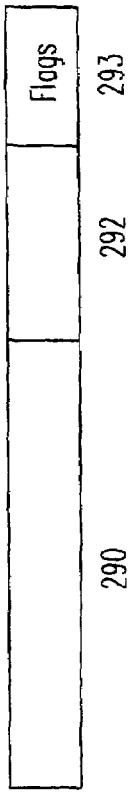

Flags
290
292
293
102

FIG. 2

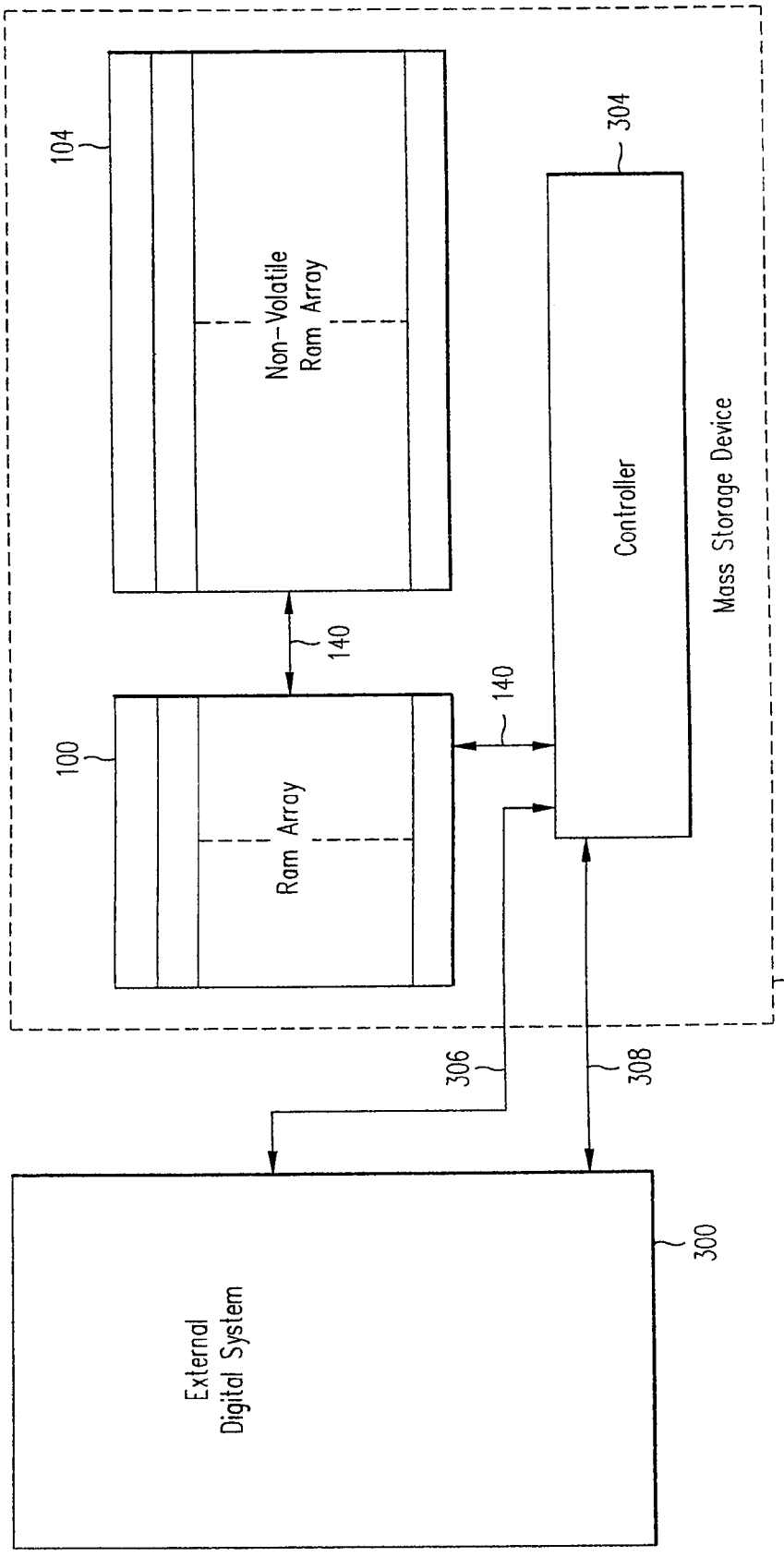

104
Non-Volatile
Ram Array
304
Controller
Mass Storage Device
140
140
100
Ram Array
302
306
308
External
Digital System
300

FIG. 3

Actual LBA/ Actual PBA	Virtual PBA	old	used	def
0		0	0	0
1		0	0	0
2		0	0	0
3		0	0	0
4		0	0	0
5		0	0	0
6		1	1	0
7		0	0	0
8		0	0	0
9		0	0	1
10		0	0	0

FIG. 4

Actual LBA/ Actual PBA	Virtual PBA	old	used	def
0		0	1	0
1		0	0	0
2		0	0	0
3	0	0	0	0
4		0	0	0
5		0	0	0
6		1	1	0
7		0	0	0
8		0	0	0
9		0	0	1
10		0	0	0

FIG. 5

Actual LBA/ Actual PBA	Virtual PBA	old	used	def
0		0	1	0
1		0	1	0
2		0	0	0
3	0	0	0	0
4	1	0	0	0
5		0	0	0
6		1	1	0
7		0	0	0
8		0	0	0
9		0	0	1
10		0	0	0

FIG. 6

Actual LBA/ Actual PBA	Virtual PBA	old	used	def
0		1	1	0
1		0	1	0
2		0	1	0
3	2	0	0	0
4	1	0	0	0
5		0	0	0
6		1	1	0
7		0	0	0
8		0	0	0
9		0	0	1
10		0	0	0

FIG. 7

Actual LBA/ Actual PBA	Virtual PBA	old	used	def
0		1	1	0
1		1	1	0
2		0	1	0
3	2	0	1	0
4	3	0	0	0
5		0	0	0
6		1	1	0
7		0	0	0
8		0	0	0
9		0	0	1
10		0	0	0

FIG. 8

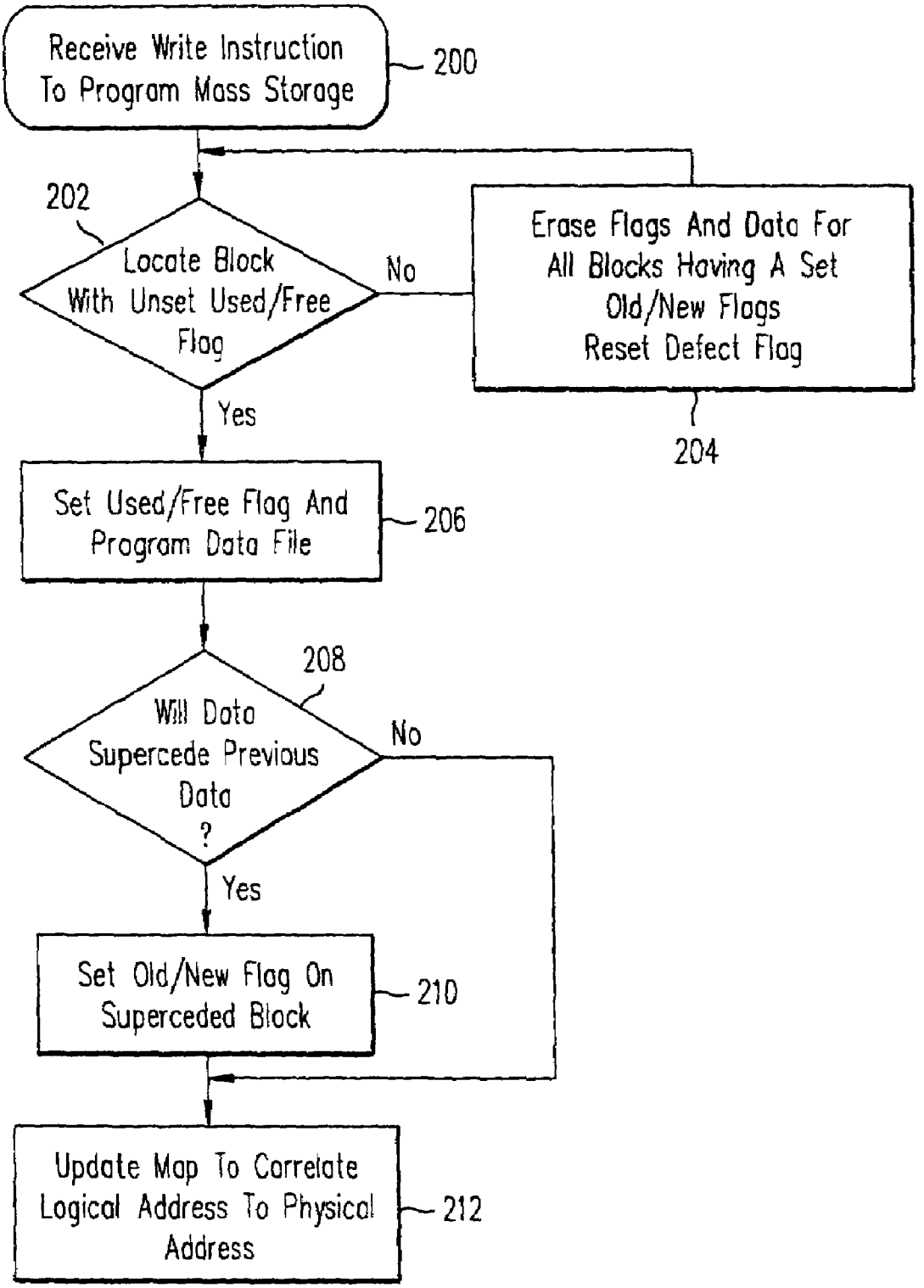

Receive Write Instruction To Program Mass Storage
200
202
Locate Block With Unset Used/Free Flag
No
Erase Flags And Data For All Blocks Having A Set Old/New Flags Reset Defect Flag
204
Yes
Set Used/Free Flag And Program Data File
206
208
Will Data Supercede Previous Data ?
No
Yes
Set Old/New Flag On Superceded Block
210
Update Map To Correlate Logical Address To Physical Address
212

FIG. 9

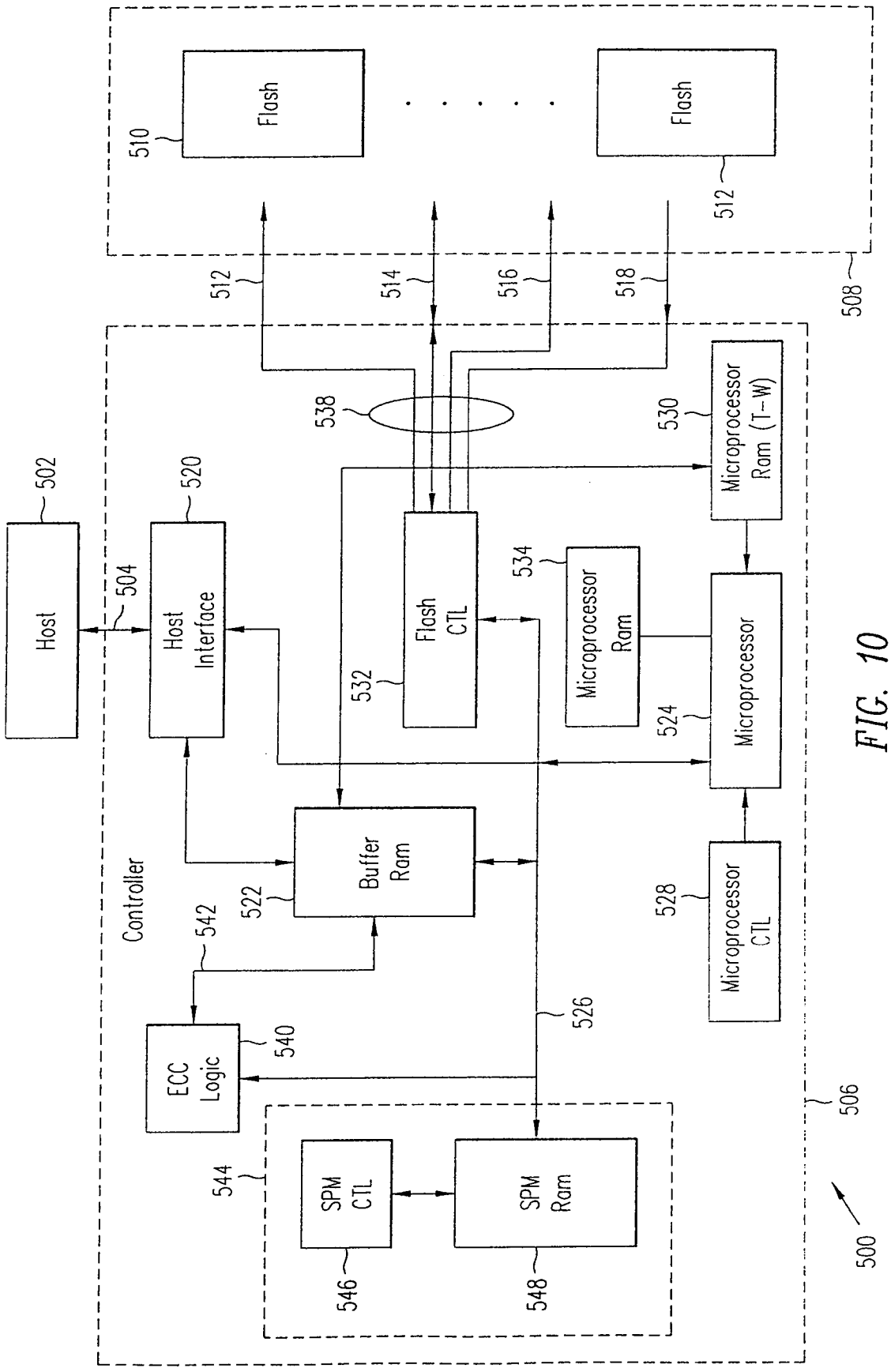

Flash
Flash
510
512
512
514
516
518
508
538
530
Microprocessor Ram (T-W)
502
520
Host
504
Host Interface
Flash CTL
534
Microprocessor Ram
532
524
Microprocessor
Buffer Ram
522
528
Microprocessor CTL
Controller
542
526
ECC Logic
540
506
544
SPM CTL
SPM Ram
546
548
500

FIG. 10

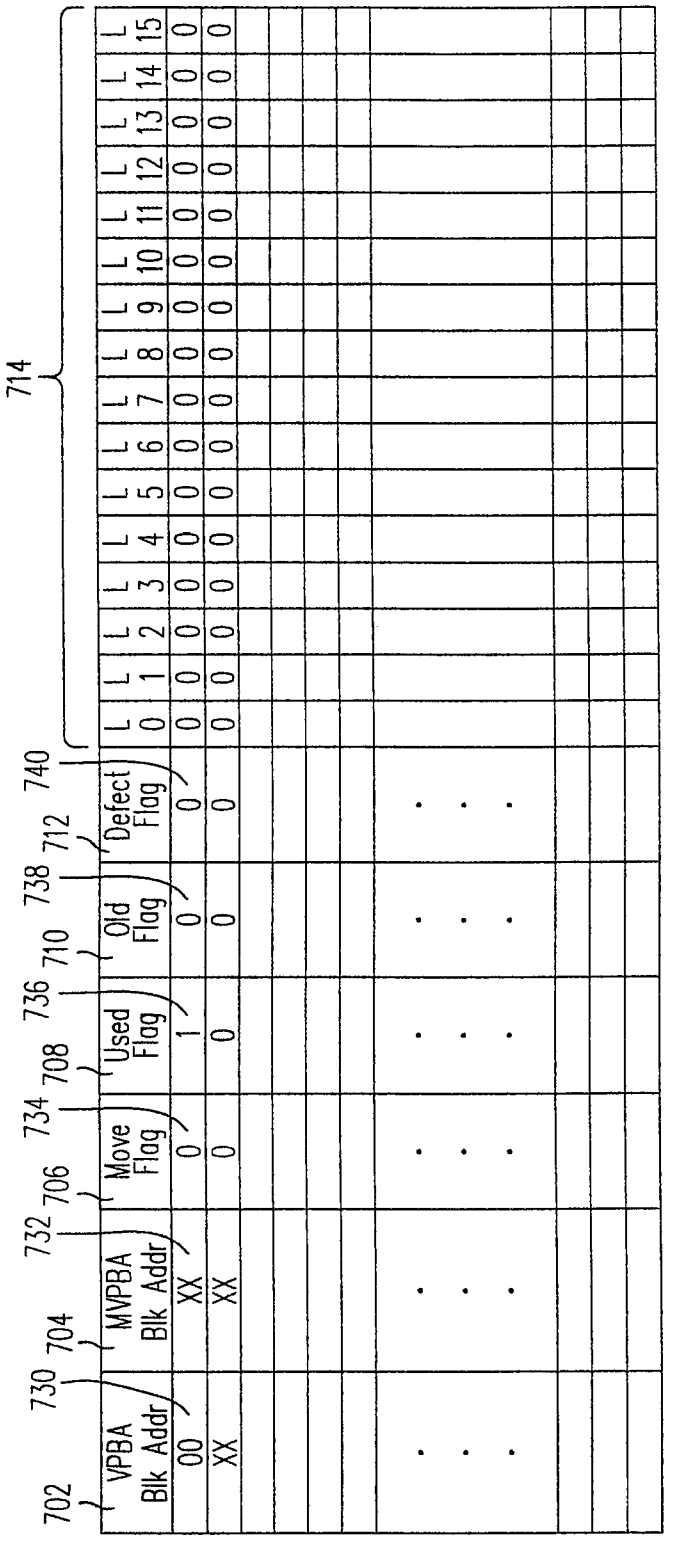

714
PBA/LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Defect Flag
L0 L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 L12 L13 L14 L15
00 XX XX 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
XX XX 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
00
10
20
30
40
50
N-1
702
704
706
708
710
712
716
718
720
722
724
726
728
730
732
734
736
738
740

FIG. 11

700

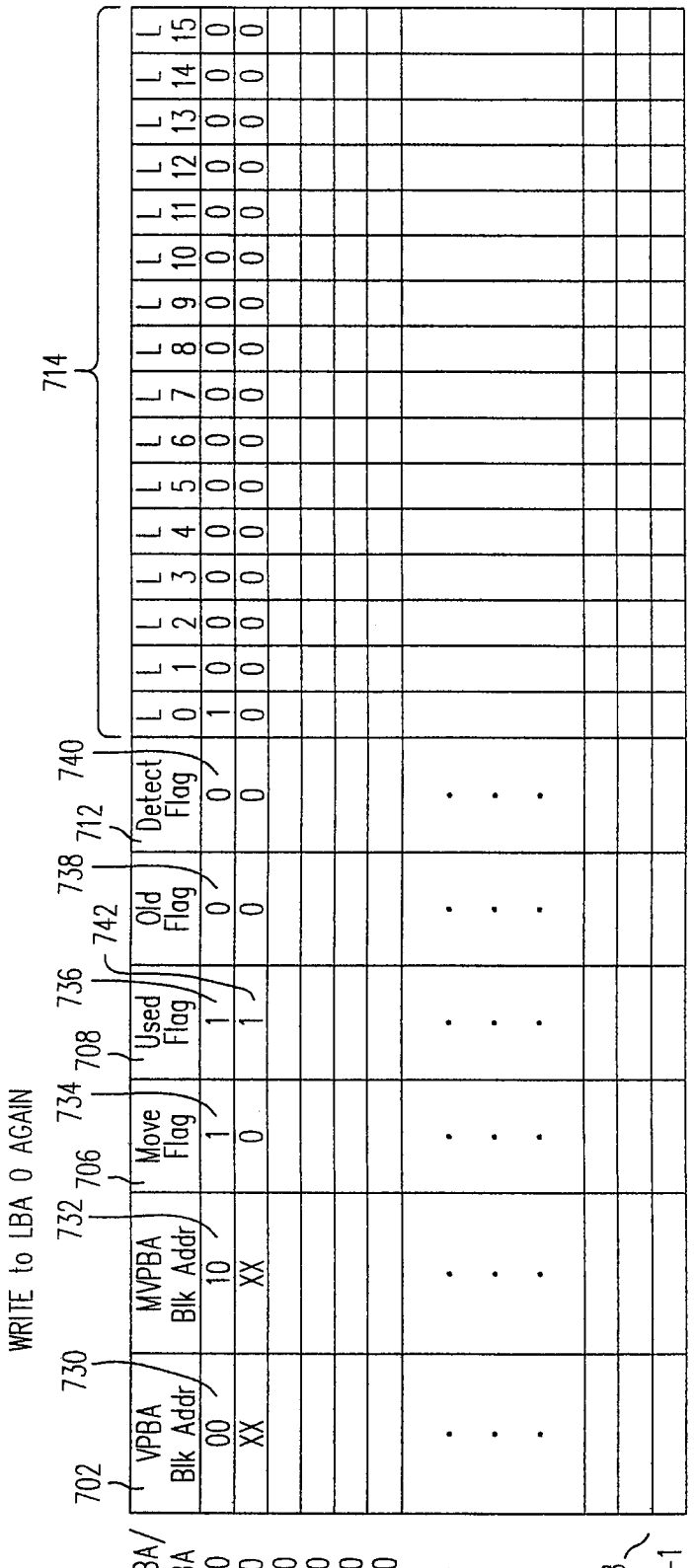
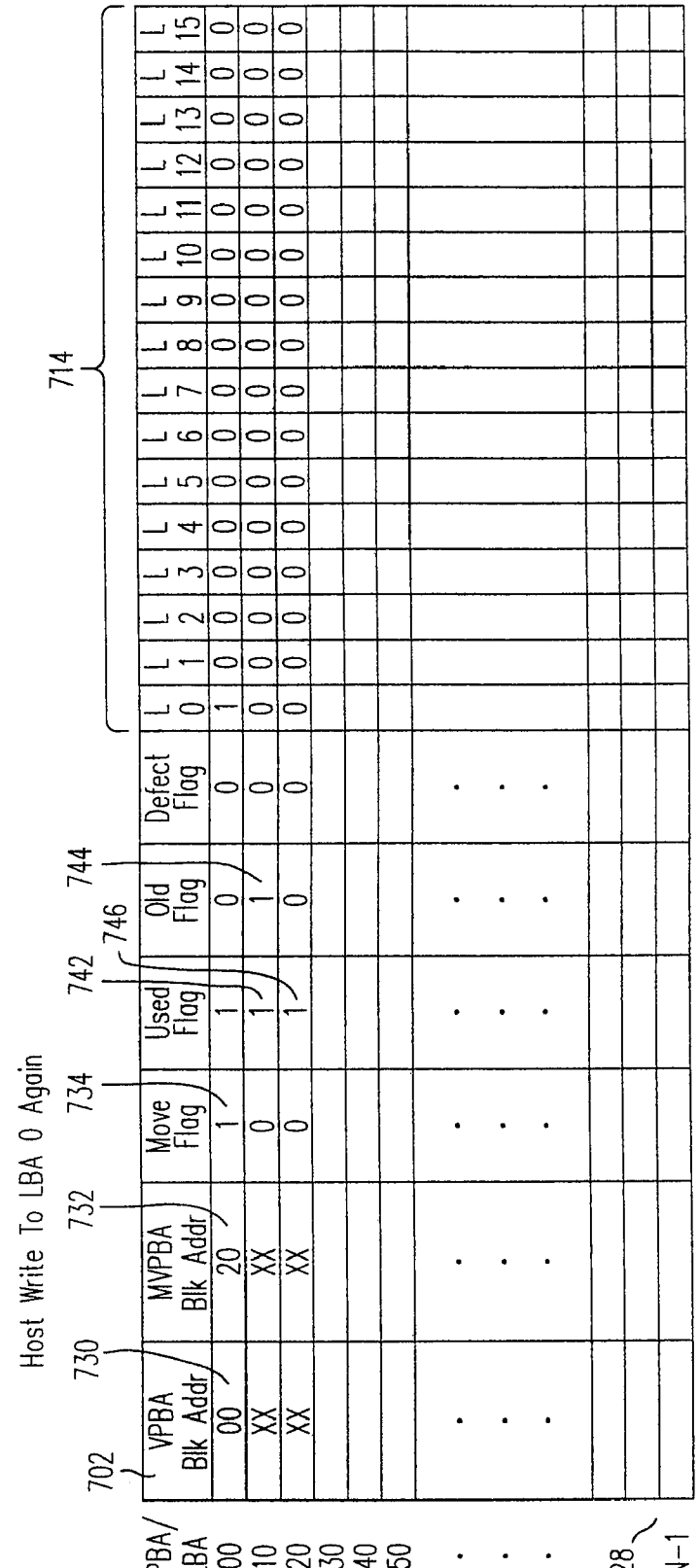

WRITE to LBA 0 AGAIN
PBA/LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Detect Flag
L 0
L 1
L 2
L 3
L 4
L 5
L 6
L 7
L 8
L 9
L 10
L 11
L 12
L 13
L 14
L 15
00
10
20
30
40
50
XX
N-1
700
702
706
708
712
714
716
718
720
722
724
726
728
730
732
734
736
738
740
742

FIG. 12


Host Write To LBA 0 Again
PBA/ LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Defect Flag
L 0
L 1
L 2
L 3
L 4
L 5
L 6
L 7
L 8
L 9
L 10
L 11
L 12
L 13
L 14
L 15
00
XX
XX
20
XX
XX
1
0
0
1
0
0
1
1
1
0
1
0
0
0
0
1
0
0
0
0

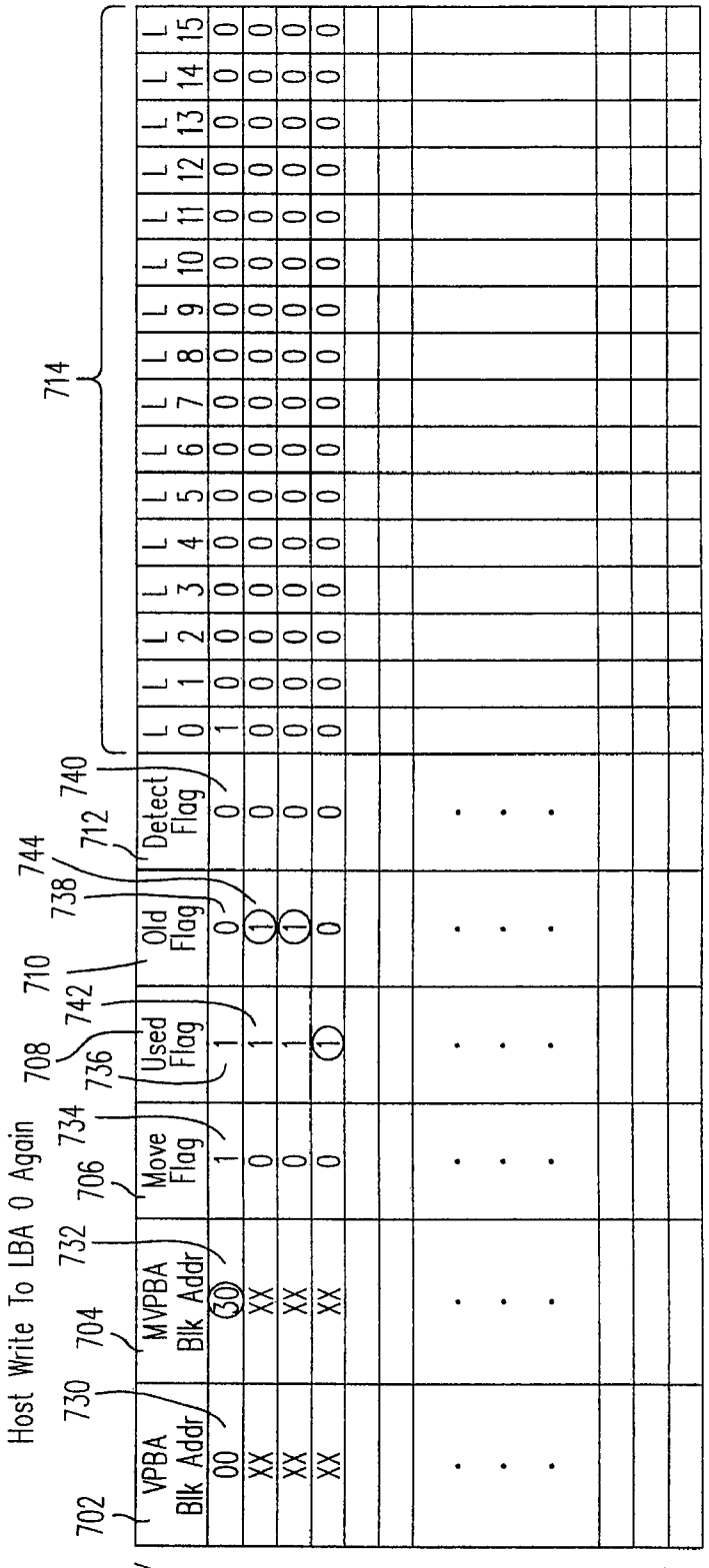

Host Write To LBA 0 Again
PBA/ LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Detect Flag
L 0
L 1
L 2
L 3
L 4
L 5
L 6
L 7
L 8
L 9
L 10
L 11
L 12
L 13
L 14
L 15
00
10
20
30
40
50
N-1
XX
700
702
704
706
708
710
712
714
716
718
720
722
724
726
728
730
732
734
736
738
740
742
744

FIG. 14

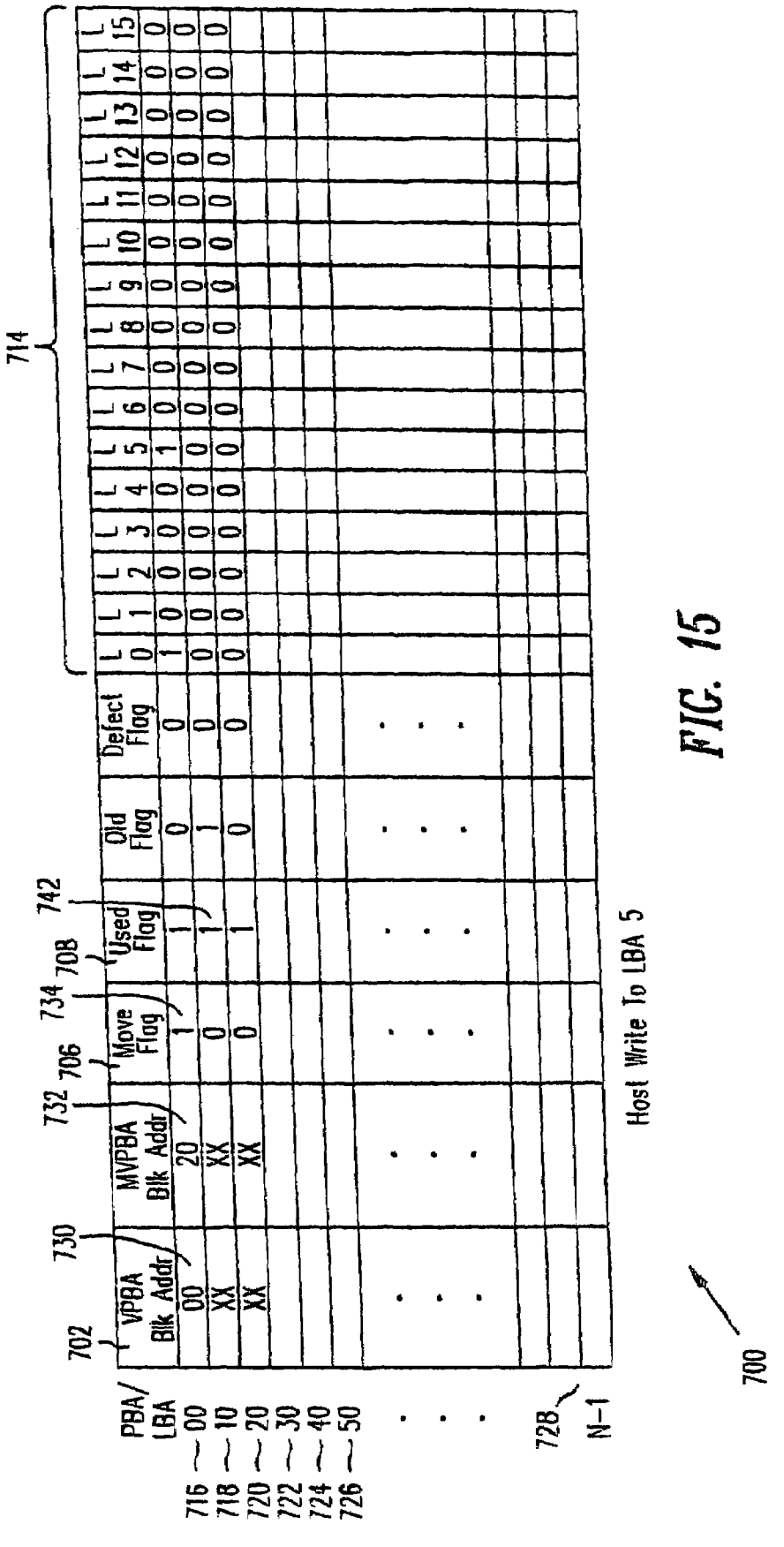

714
PBA/LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Defect Flag
L0 L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 L12 L13 L14 L15
00 20 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
10 XX 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
20 XX 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
30
40
50
N-1
716
718
720
722
724
726
728
730
732
734
706
708
742
702
Host Write To LBA 5
700

FIG. 15

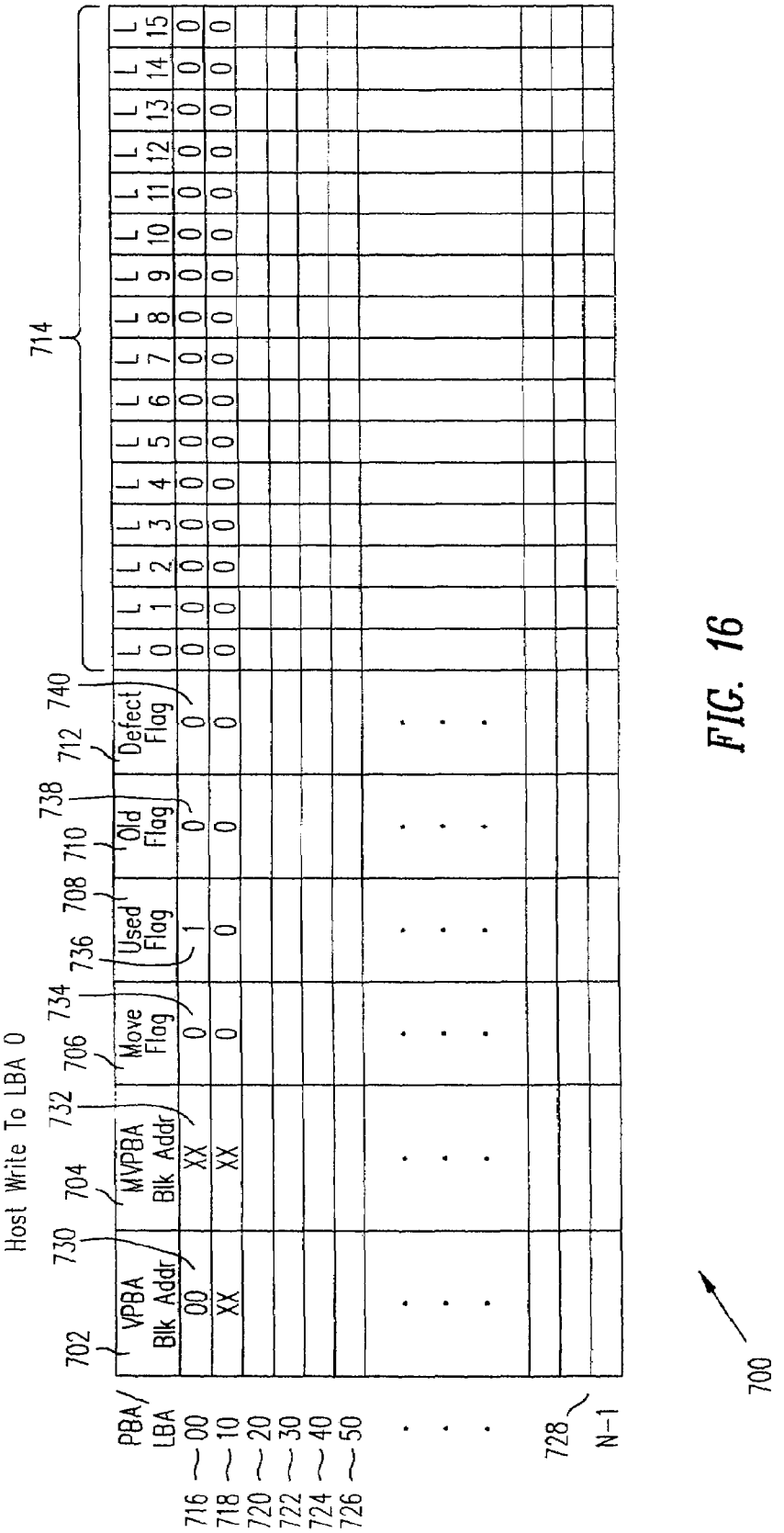
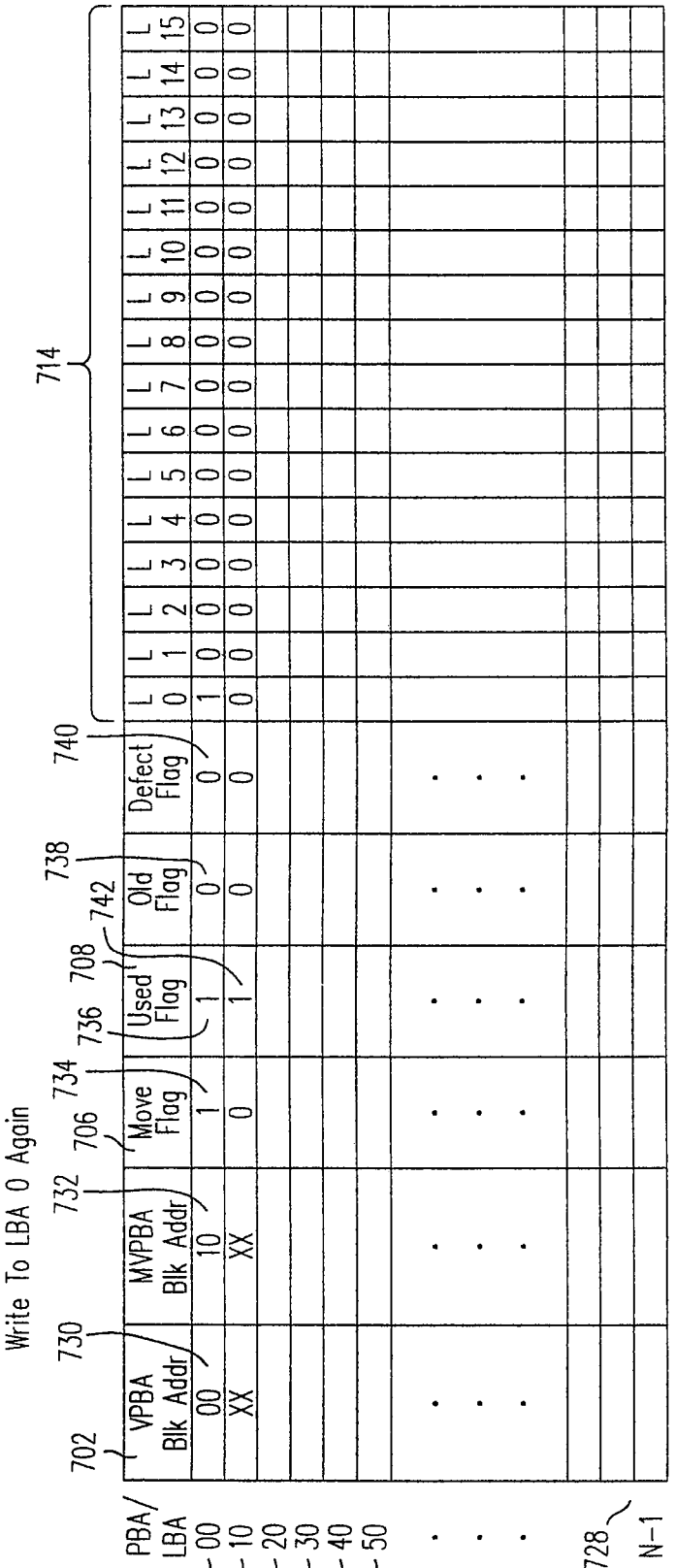
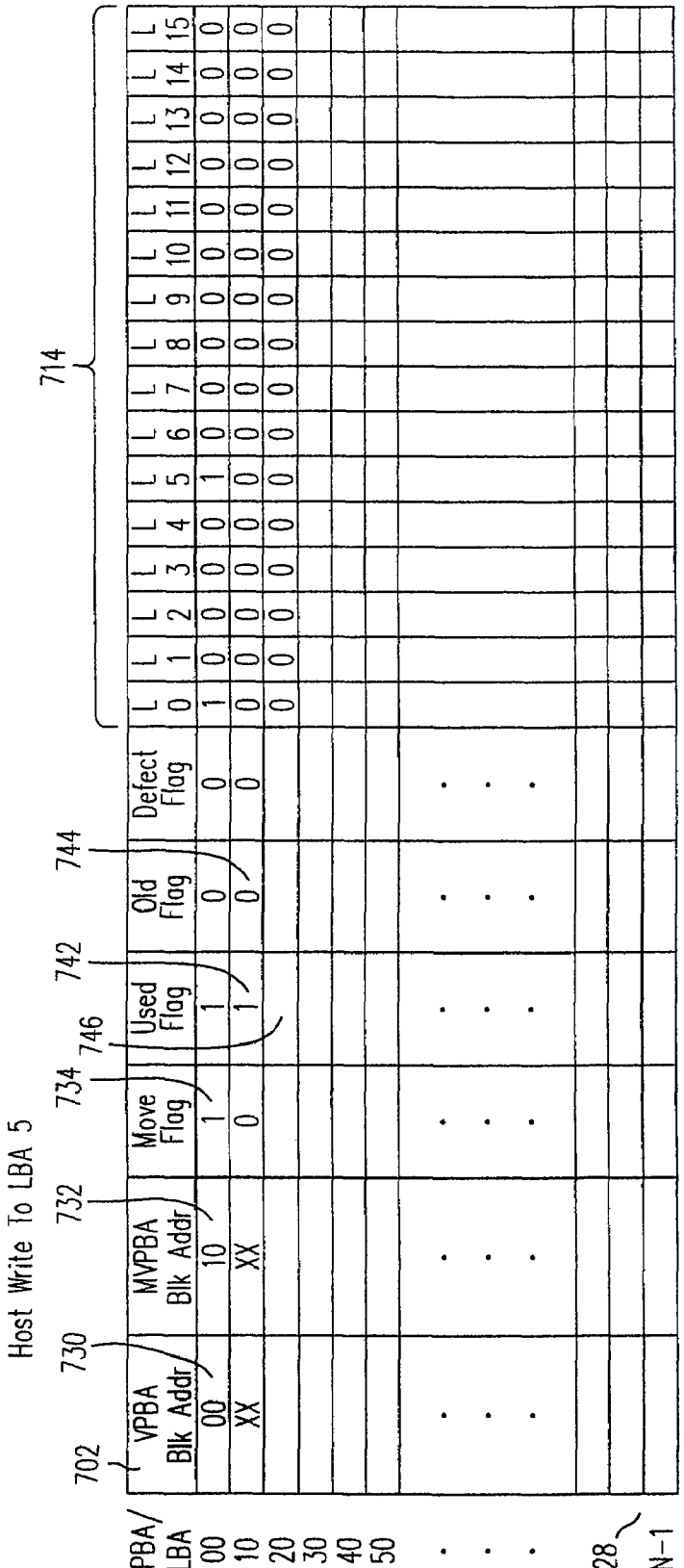

Host Write To LBA 0
PBA/LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Defect Flag
L 0
L 1
L 2
L 3
L 4
L 5
L 6
L 7
L 8
L 9
L 10
L 11
L 12
L 13
L 14
L 15
00
10
20
30
40
50
N-1
XX
700
702
704
706
708
710
712
714
716
718
720
722
724
726
728
730
732
734
736
738
740

FIG. 16


Write To LBA 0 Again
PBA/LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Defect Flag
L 0
L 1
L 2
L 3
L 4
L 5
L 6
L 7
L 8
L 9
L 10
L 11
L 12
L 13
L 14
L 15
00
10
20
30
40
50
N-1
00
10
XX
XX
0
1
1
0
0
0


Host Write To LBA 5
PBA/LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Defect Flag
L 0
L 1
L 2
L 3
L 4
L 5
L 6
L 7
L 8
L 9
L 10
L 11
L 12
L 13
L 14
L 15
00
10
XX
XX
0
1
1
0
0
0

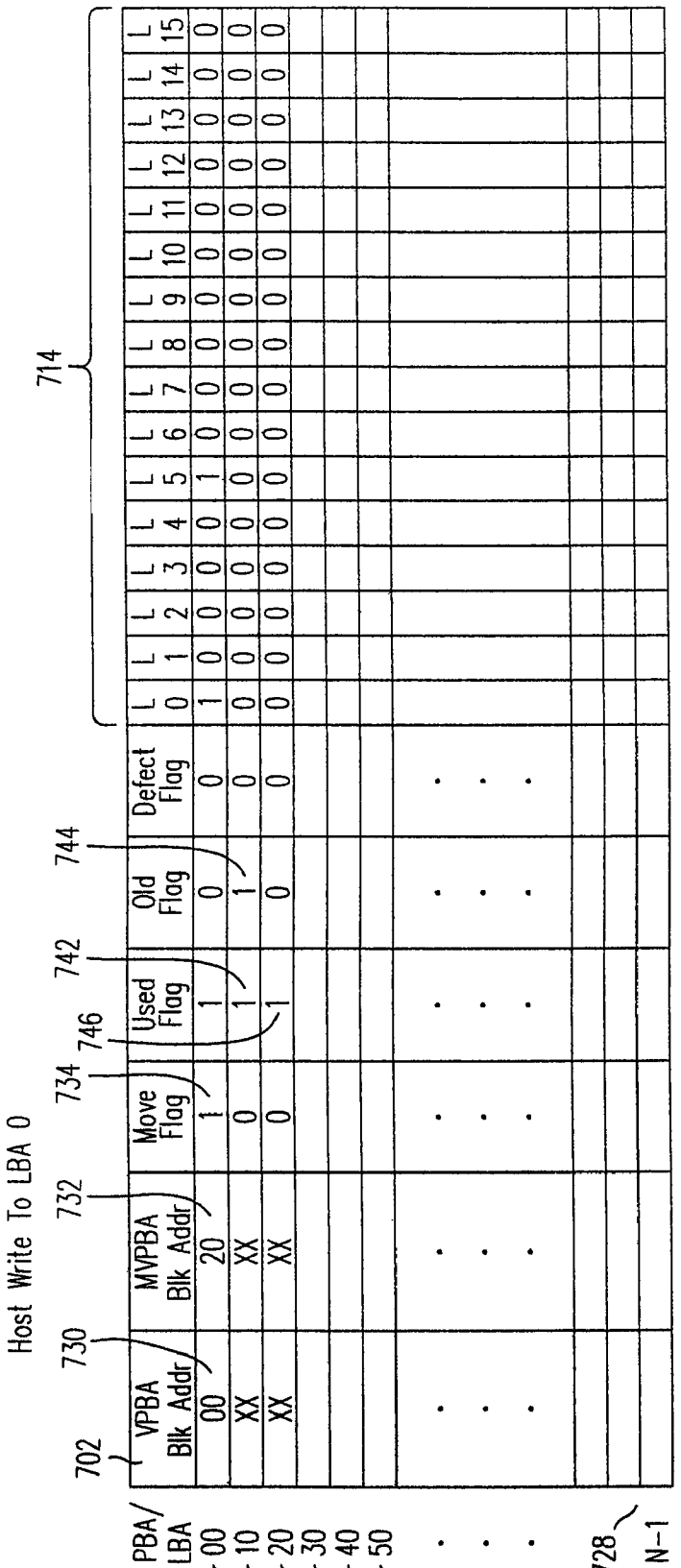

Host Write To LBA 0
PBA/LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Defect Flag
L 0
L 1
L 2
L 3
L 4
L 5
L 6
L 7
L 8
L 9
L 10
L 11
L 12
L 13
L 14
L 15
00
10
20
30
40
50
N-1
702
730
732
734
742
744
746
714
716
718
720
722
724
726
728
700

FIG. 19

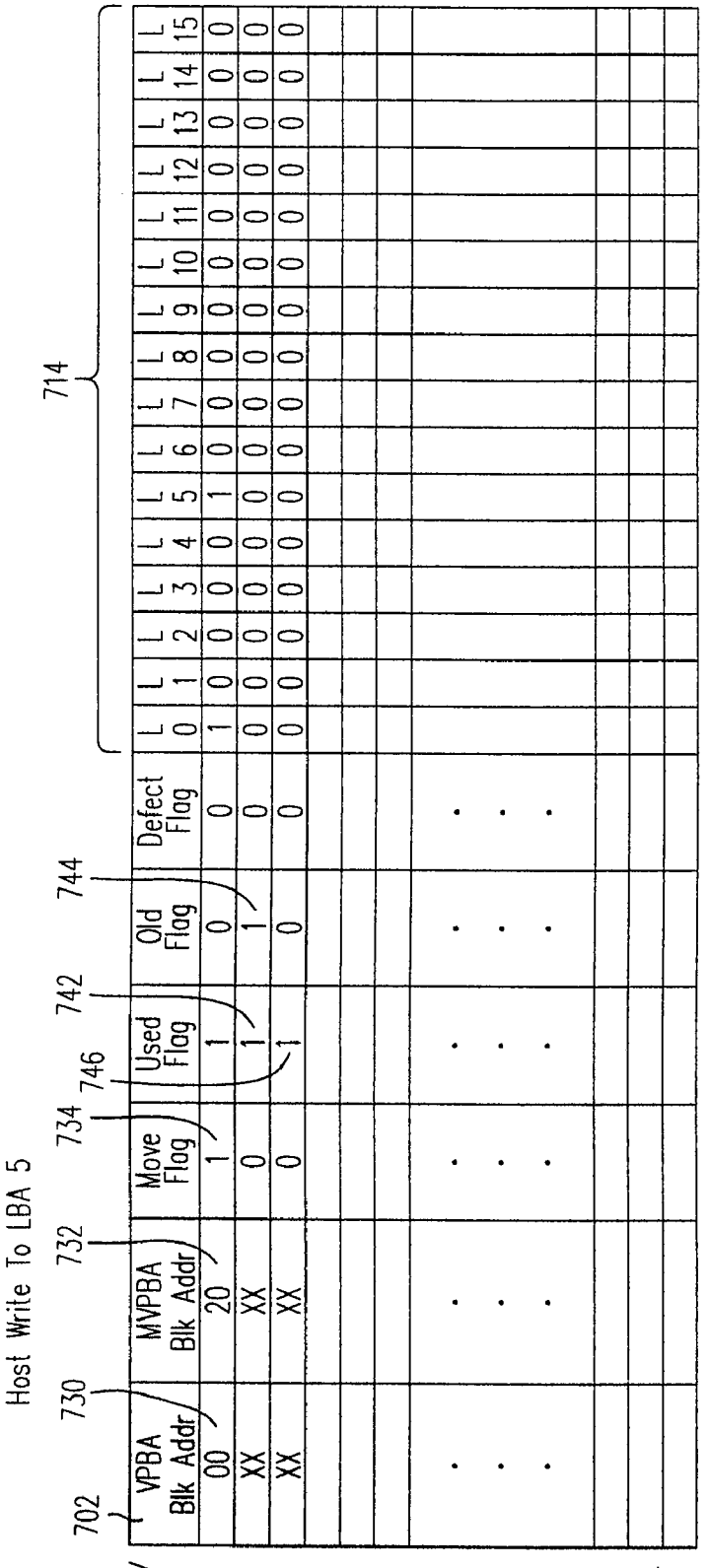

Host Write To LBA 5
PBA/ LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Defect Flag
L 0
L 1
L 2
L 3
L 4
L 5
L 6
L 7
L 8
L 9
L 10
L 11
L 12
L 13
L 14
L 15
00
10
20
30
40
50
N-1
XX
20
714
730
732
734
742
744
746
716
718
720
722
724
726
728
700
702

FIG. 20

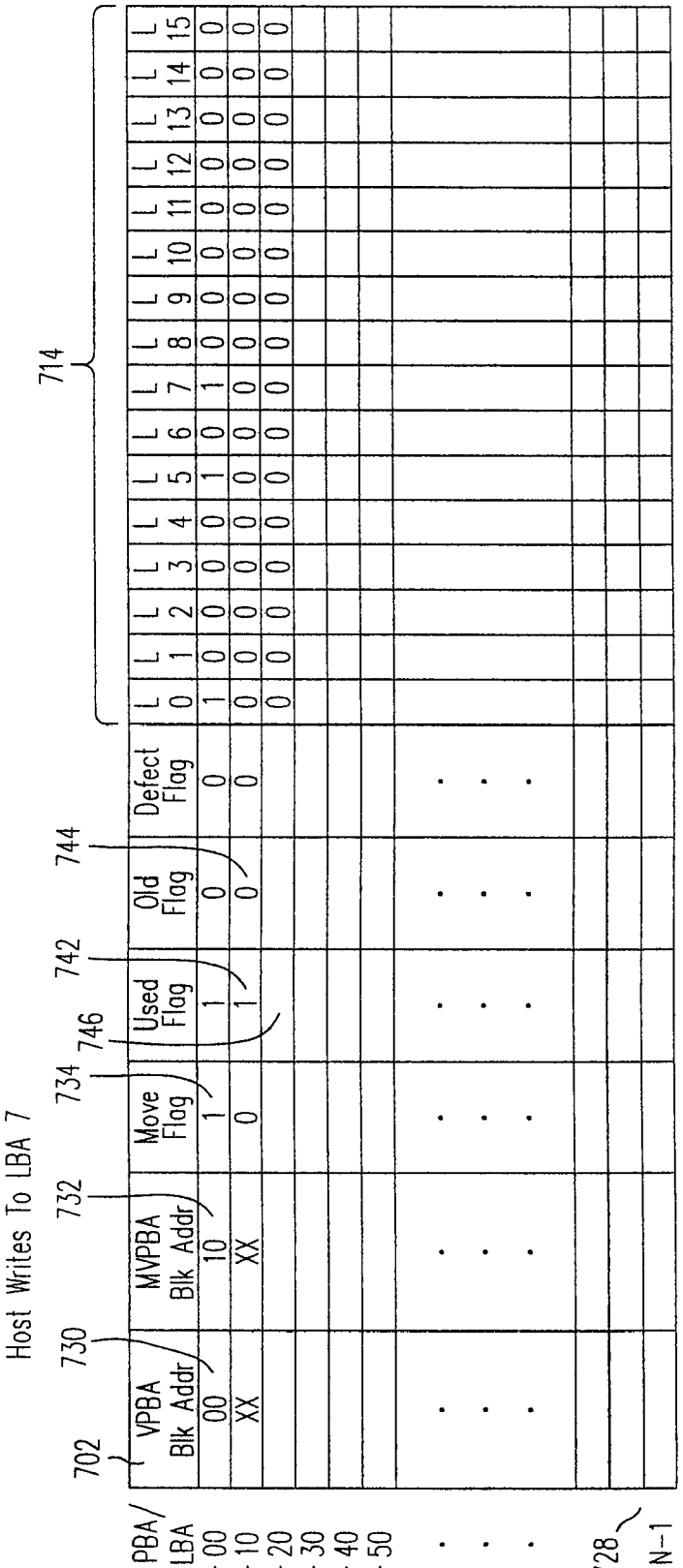

Host Writes To LBA 7
PBA/LBA
VPBA Blk Addr
MVPBA Blk Addr
Move Flag
Used Flag
Old Flag
Defect Flag
L0 L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 L12 L13 L14 L15
00 10 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0
XX XX 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
00
10
20
30
40
50
N-1
702
730
732
734
742
744
746
714
716
718
720
722
724
726
728
700

FIG. 21

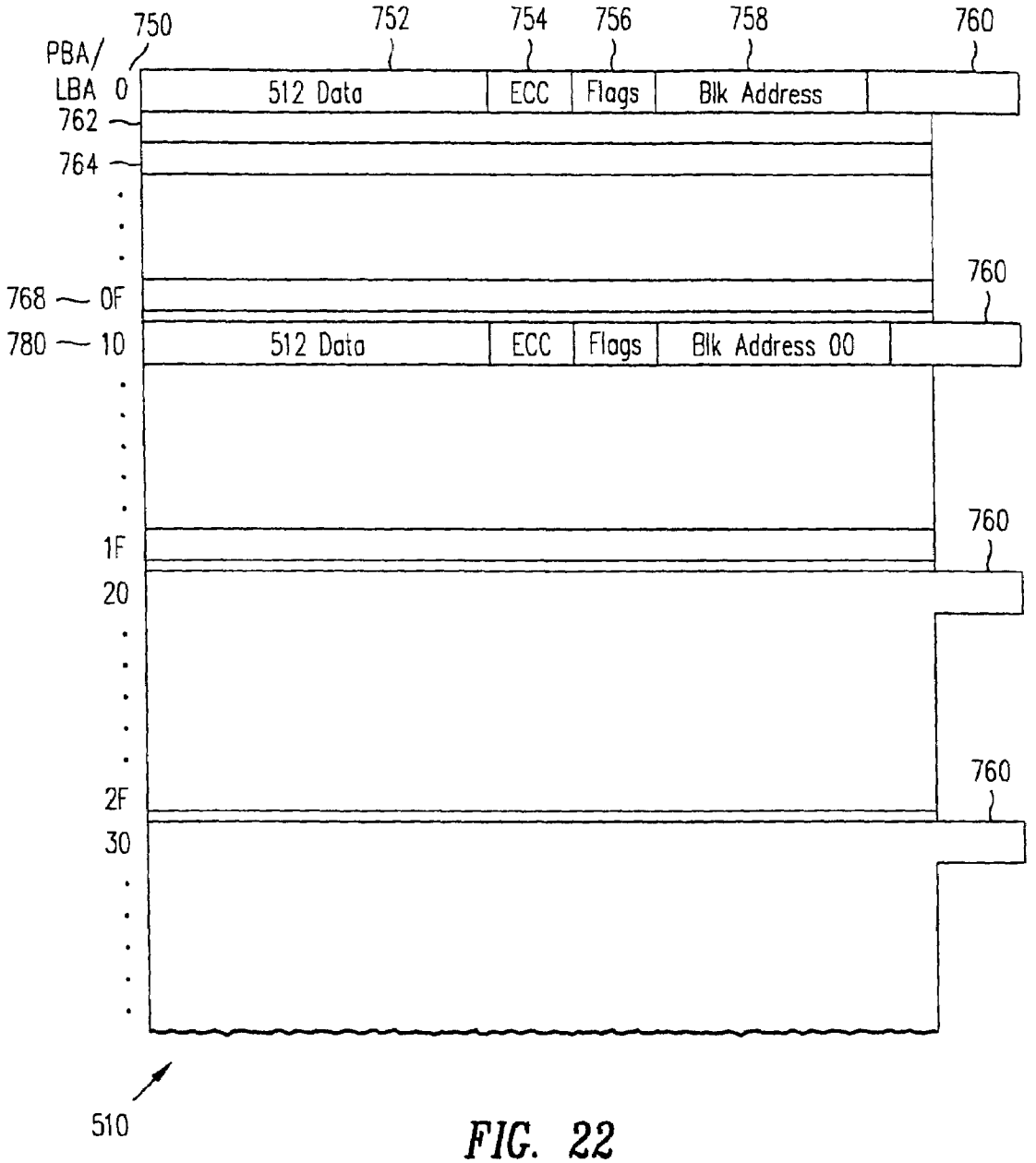

750
752
754
756
758
760
PBA/
LBA 0
512 Data
ECC
Flags
Blk Address
762
764
768 — 0F
780 — 10
512 Data
ECC
Flags
Blk Address 00
1F
20
2F
30
510

FIG. 22

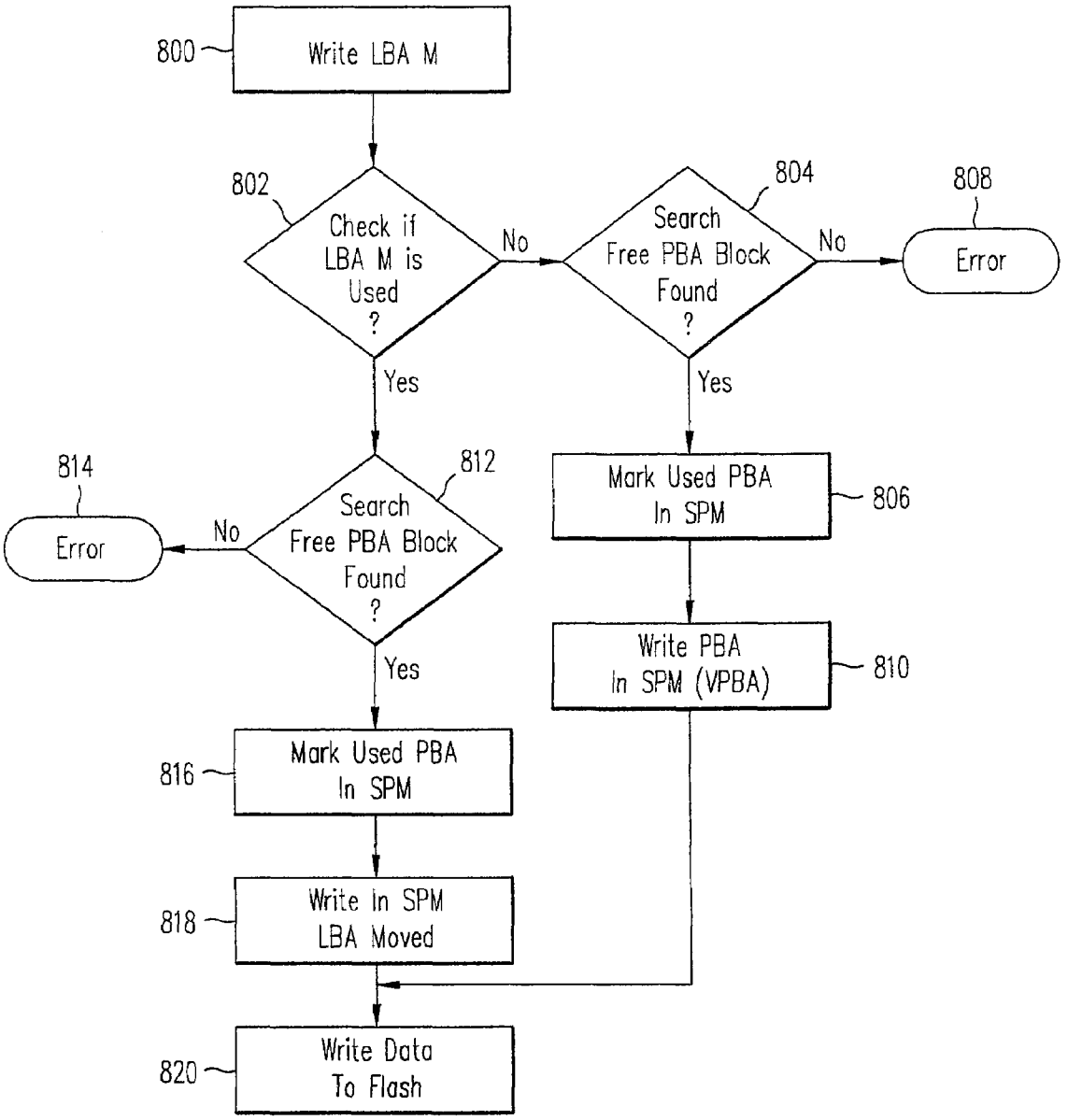

800
Write LBA M
802
Check if LBA M is Used ?
No
804
Search Free PBA Block Found ?
No
808
Error
Yes
Yes
814
Error
No
812
Search Free PBA Block Found ?
Mark Used PBA In SPM
806
Write PBA In SPM (VPBA)
810
Yes
816
Mark Used PBA In SPM
818
Write In SPM LBA Moved
820
Write Data To Flash

FIG. 23

SYSTEM AND METHOD FOR UPDATING DATA SECTORS IN A NON-VOLATILE MEMORY USING LOGICAL BLOCK ADDRESSING

CROSS REFERENCE TO RELATED APPLICATION

This application is a continuation of a prior U.S. application Ser. No. 09/620,544, filed on Jul. 21, 2000, now U.S. Pat. No. 6,978,342, which is a continuation of U.S. Ser. No. 09/264,340, filed on Mar. 8, 1999, now U.S. Pat. No. 6,145,051, and entitled "Moving Sectors Within A Block of Information In A Flash Memory Mass Storage Architecture," which is a continuation of a prior application Ser. No. 08/831,266, filed on Mar. 31, 1997, now U.S. Pat. No. 5,907,856, and entitled "Moving Sectors Within A Block of Information In A Flash Memory Mass Storage Architecture," which is a continuation-in-part of prior application Ser. No. 08/509,706, filed on Jul. 31, 1995, now U.S. Pat. No. 5,845,313, and entitled "Direct Logical Block Addressing Flash Memory Mass Storage Architecture." Application Nos. 08/831,266 and 08/509,706 are incorporated herein by reference.

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to the field of mass storage for computers. More particularly, this invention relates to an architecture for replacing a hard disk with a semiconductor nonvolatile memory and in particular flash memory.

Computers conventionally use rotating magnetic media for mass storage of documents, data, programs and information. Though widely used and commonly accepted, such hard disk drives suffer from a variety of deficiencies. Because of the rotation of the disk, there is an inherent latency in extracting information from a hard disk drive.

Other problems are especially dramatic in portable computers. In particular, hard disks are unable to withstand many of the kinds of physical shock that a portable computer will likely sustain. Further, the motor for rotating the disk consumes significant amounts of power decreasing the battery life for portable computers.

Solid state memory is an ideal choice for replacing a hard disk drive for mass storage because it can resolve the problems cited above. Potential solutions have been proposed for replacing a hard disk drive with a semiconductor memory. For such a system to be truly useful, the memory must be nonvolatile and alterable. The inventors have determined that FLASH memory is preferred for such a replacement.

FLASH memory is a transistor memory cell which is programmable through hot electron, source injection, or tunneling, and erasable through Fowler-Nordheim tunneling. The programming and erasing of such a memory cell requires current to pass through the dielectric surrounding floating gate electrode. Because of this, such types of memory have a finite number of erase-write cycles. Eventually, the dielectric deteriorates. Manufacturers of FLASH cell devices specify the limit for the number of erase-write cycles between 100,000 and 1,000,000.

One requirement for a semiconductor mass storage device to be successful is that its use in lieu of a rotating media hard disk mass storage device be transparent to the designer and the user of a system using such a device. In other words, the designer or user of a computer incorporating such a semiconductor mass storage device could simply remove the hard disk and replace it with a semiconductor mass storage device. All presently available commercial software should operate on a system employing such a semiconductor mass storage device without the necessity of any modification.

SanDisk proposed an architecture for a semiconductor mass storage using FLASH memory at the Silicon Valley PC Design Conference on Jul. 9, 1991. That mass storage system included read-write block sizes of 512 Bytes to conform with commercial hard disk sector sizes.

Earlier designs incorporated erase-before-write architectures. In this process, in order to update a file on the media, if the physical location on the media was previously programmed, it has to be erased before the new data can be reprogrammed.

This process would have a major deterioration on overall system throughput. When a host writes a new data file to the storage media, it provides a logical block address to the peripheral storage device associated with this data file. The storage device then translates this given logical block address to an actual physical block address on the media and performs the write operation. In magnetic hard disk drives, the new data can be written over the previous old data with no modification to the media. Therefore, once the physical block address is calculated from the given logical block address by the controller, it will simply write the data file into that location. In solid state storage, if the location associated with the calculated physical block address was previously programmed, before this block can be reprogrammed with the new data, it has to be erased. In one previous art, in erase-before-write architecture where the correlation between logical block address given by the host is one to one mapping with physical block address on the media. This method has many deficiencies. First, it introduces a delay in performance due to the erase operation before reprogramming the altered information. In solid state flash, erase is a very slow process.

Secondly, hard disk users typically store two types of information, one is rarely modified and another which is frequently changed. For example, a commercial spread sheet or word processing software program stored on a user's system are rarely, if ever, changed. However, the spread sheet data files or word processing documents are frequently changed. Thus, different sectors of a hard disk typically have dramatically different usage in terms of the number of times the information stored thereon is changed. While this disparity has no impact on a hard disk because of its insensitivity to data changes, in a FLASH memory device, this variance can cause sections of the mass storage to wear out and be unusable significantly sooner than other sections of the mass storage.

In another architecture, the inventors previously proposed a solution to store a table correlating the logical block address to the physical block address. The inventions relating to that solution are disclosed in U.S. patent application Ser. No. 08/038,668 filed on Mar. 26, 1993, now U.S. Pat. No. 5,388,083, and U.S. patent application Ser. No. 08/037,893 also filed on Mar. 26, 1993, now U.S. Pat. No. 5,479,638. Those applications are incorporated herein by reference.

The inventors' previous solution discloses two primary algorithms and an associated hardware architecture for a semiconductor mass storage device. It will be understood that "data file" in this patent document refers to any computer file including commercial software, a user program, word processing software document, spread sheet file and the like. The first algorithm in the previous solution provides means for avoiding an erase operation when writing a modified data file back onto the mass storage device. Instead, no erase is performed and the modified data file is written onto an empty portion of the mass storage.

The semiconductor mass storage architecture has blocks sized to conform with commercial hard disk sector sizes. The blocks are individually erasable. In one embodiment, the semiconductor mass storage can be substituted for a rotating hard disk with no impact to the user, so that such a substitution will be transparent. Means are provided for avoiding the erase-before-write cycle each time information stored in the mass storage is changed.

According to the first algorithm, erase cycles are avoided by programming an altered data file into an empty block. This would ordinarily not be possible when using conventional mass storage because the central processor and commercial software available in conventional computer systems are not configured to track continually changing physical locations of data files. The previous solution includes a programmable map to maintain a correlation between the logical address and the physical address of the updated information files.

All the flags, and the table correlating the logical block address to the physical block address are maintained within an array of CAM cells. The use of the CAM cells provides very rapid determination of the physical address desired within the mass storage, generally within one or two clock cycles. Unfortunately, as is well known, CAM cells require multiple transistors, typically six. Accordingly, an integrated circuit built for a particular size memory using CAM storage for the tables and flags will need to be significantly larger than a circuit using other means for just storing the memory.

The inventors proposed another solution to this problem which is disclosed in U.S. patent application Ser. No. 08/131,495 filed on Oct. 4, 1993, now U.S. Pat. No. 5,485,595. That application is incorporated herein by reference.

This additional previous solution invented by these same inventors is also for a nonvolatile memory storage device. The device is also configured to avoid having to perform an erase-before-write each time a data file is changed by keeping a correlation between logical block address and physical block address in a volatile space management RAM. Further, this invention avoids the overhead associated with CAM cell approaches which require additional circuitry.

Like the solutions disclosed above by these same inventors, the device includes circuitry for performing the two primary algorithms and an associated hardware architecture for a semiconductor mass storage device. In addition, the CAM cell is avoided in this previous solution by using RAM cells.

Reading is performed in this previous solutions by providing the logical block address to the memory storage. The system sequentially compares the stored logical block addresses until it finds a match. That data file is then coupled to the digital system. Accordingly, the performance offered by this solution suffers because potentially all of the memory locations must be searched and compared to the desired logical block address before the physical location of the desired information can be determined.

What is needed is a semiconductor hard disk architecture which provides rapid access to stored data without the excessive overhead of CAM cell storage.

SUMMARY OF THE INVENTION

The present invention is for a nonvolatile memory storage device. The device is configured to avoid having to perform an erase-before-write each time a data file is changed. Further, to avoid the overhead associated with CAM cells, this approach utilizes a RAM array. The host system maintains organization of the mass storage data by using a logical block address. The RAM array is arranged to be addressable by the same address as the logical block addresses (LBA) of the host. Each such addressable location in the RAM includes a field which holds the physical address of the data in the nonvolatile mass storage expected by the host. This physical block address (PBA) information must be shadowed in the nonvolatile memory to ensure that the device will still function after resuming operation after a power down because RAMs are volatile memory devices. In addition, status flags are also stored for each physical location. The status flags can be stored in either the nonvolatile media or in both the RAM and in the nonvolatile media.

The device includes circuitry for performing two primary algorithms and an associated hardware architecture for a semiconductor mass storage device. The first algorithm provides a means for mapping of host logical block address to physical block address with much improved performance and minimal hardware assists. In addition, the second algorithm provides means for avoiding an erase-before-write cycle when writing a modified data file back onto the mass storage device. Instead, no erase is performed and the modified data file is written onto an empty portion of the mass storage.

Reading is performed in the present invention by providing the logical block address to the memory storage. The RAM array is arranged so that the logical block address selects one RAM location. That location contains the physical block address of the data requested by the host or other external system. That data file is then read out to the host.

According to the second algorithm, erase cycles are avoided by programming an altered data file into an altered data mass storage block rather than itself after an erase cycle of the block as done on previous arts.

In an alternative embodiment of the present invention, a method and apparatus is presented for efficiently moving sectors within a block from a first area within the nonvolatile memory to an unused area within the nonvolatile memory and marking the first area as "used".

Briefly, a preferred embodiment of the present invention includes a method and apparatus for storing mapping information for mapping a logical block address identifying a block being accessed by a host to a physical block address, identifying a free area of nonvolatile memory, the block being selectively erasable and having one or more sectors that may be individually moved. The mapping information including a virtual physical block address for identifying an "original" location, within the nonvolatile memory, wherein a block is stored and a moved virtual physical block address for identifying a "moved" location, within the nonvolatile memory, wherein one or more sectors of the stored block are moved. The mapping information further including status information for use of the "original" physical block address and the "moved" physical block address and for providing information regarding "moved" sectors within the block being accessed.

IN THE DRAWINGS

FIG. **1** shows a schematic block diagram of an architecture for a semiconductor mass storage according to the present invention.

FIG. **2** shows an alternative embodiment to the physical block address **102** of the RAM storage of FIG. **1**.

FIG. **3** shows a block diagram of a system incorporating the mass storage device of the present invention.

FIGS. **4** through **8** show the status of several of the flags and information for achieving the advantages of the present invention.

FIG. **9** shows a flow chart block diagram of the first algorithm according to the present invention.

FIG. **10** shows a high-level block diagram of a digital system, such as a digital camera, including a preferred embodiment of the present invention.

FIGS. **11-21** illustrate several examples of the state of a mapping table that may be stored in the digital system of FIG. **10** including LBA-PBA mapping information.

FIG. **22** depicts an example of a nonvolatile memory device employed in the preferred embodiment of FIG. **10**.

FIG. **23** shows a high-level flow chart of the general steps employed in writing a block of information to the nonvolatile devices of FIG. **10**.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. **1** shows an architecture for implementation of a solid state storage media according to the present invention. The storage media is for use with a host or other external digital system. The mass storage is partitioned into two portions, a volatile RAM array **100** and a nonvolatile array **104**. According to the preferred embodiment, all of the nonvolatile memory storage is FLASH. The FLASH may be replaced by EEPROM. The RAM can be of any convenient type.

The memory storage **104** is arranged into N blocks of data from zero through N−1. Each of the blocks of data is M Bytes long. In the preferred embodiment, each data block is 512 Bytes long to correspond with a sector length in a commercially available hard disk drive plus the extra numbers of bytes to store the flags and logical block address (LBA) information and the associated ECC. The memory **104** can contain as much memory storage as a user desires. An example of a mass storage device might include 100 M Byte of addressable storage.

There are a plurality of RAM locations **102**. Each RAM location **102** is uniquely addressable by controller using an appropriate one of the logical block addresses provided by the host system or the actual physical address of the nonvolatile media. The RAM location **102** contains the physical block address of the data associated with the logical block address and the flags associated with a physical block address on the nonvolatile media.

It is possible that the physical block address (PBA) can be split into two fields as shown in FIG. **2**. These fields can be used for cluster addresses of a group of data blocks. The first such field **290** is used to select a cluster address and the second such field **292** can be used to select the start address of the logical block address associated with this cluster.

A collection of information flags is also stored for each nonvolatile memory location **106**. These flags include an old/new flag **110**, a used/free flag **112**, a defect flag **114**, and a single/sector flag **116**. Additionally, there is also a data store **122**.

When writing data to the mass storage device of the present invention, a controller determines the first available physical block for storing the data. The RAM location **102** corresponding to the logical block address selected by the host is written with the physical block address where the data is actually stored within the nonvolatile memory array in **104** (FIG. **1**).

Assume for example that a user is preparing a word processing document and instructs the computer to save the document. The document will be stored in the mass storage system. The host system will assign it a logical block address. The mass storage system of the present invention will select a physical address of an unused block or blocks in the mass storage for storing the document. The address of the physical block address will be stored into the RAM location **102** corresponding to the logical block address. As the data is programmed, the system of the present invention also sets the used free flag **112** in **104** and **293** to indicate that this block location is used. One used/free flag **112** is provided for each entry of the nonvolatile array **104**.

Later, assume the user retrieves the document, makes a change and again instructs the computer to store the document. To avoid an erase-before-write cycle, the system of the present invention provides means for locating a block having its used/free flag **112** in **100** unset (not programmed) which indicates that the associated block is erased. The system then sets the used/free flag for the new block **112** of **106** and **293** of **100** and then stores the modified document in that new physical block location **106** in the nonvolatile array **104**. The address of the new physical block location is also stored into the RAM location **102** corresponding the logical block address, thereby writing over the previous physical block location in **102**. Next, the system sets the old/new flag **110** of the previous version of the document indicating that this is an old unneeded version of the document in **110** of **104** and **293** of **109**. In this way, the system of the present invention avoids the overhead of an erase cycle which is required in the erase-before-write of conventional systems to store a modified version of a previous document.

Because of RAM array **100** will lose its memory upon a power down condition, the logical block address with the active physical block address in the media is also stored as a shadow memory **108** in the nonvolatile array **104**. It will be understood the shadow information will be stored into the appropriate RAM locations **102** by the controller. During power up sequence, the RAM locations in **100** are appropriately updated from every physical locations in **104**, by reading the information **106** of **104**. The logical address **108** of **106** is used to address the RAM location of **100** to update the actual physical block address associated with the given logical block address. Also since **106** is the actual physical block address associated with the new data **122**, the flags **110**, **112**; **114**, and **116** arc updated in **293** of **102** with the physical block address of **106** in **100**. It will be apparent to one of ordinary skill in the art that the flags can be stored in either the appropriate nonvolatile memory location **106** or in both the nonvolatile memory location and also in the RAM location **102** associated with the physical block address.

During power up, in order to assign the most recent physical block address assigned to a logical block address in the volatile memory **100**, the controller will first read the Flags **110**, **112**, **114**, and **116** portion of the nonvolatile memory **104** and updates the flags portion **293** in the volatile memory **100**. Then it reads the logical block address **108** of every physical block address of the nonvolatile media **104** and by tracking the flags of the given physical block address in the volatile memory **100**, and the read logical block address of the physical block address in the nonvolatile memory **104**, it can update the most recent physical block address assigned to the read logical block address in the volatile memory **100**.

FIG. **3** shows a block diagram of a system incorporating the mass storage device of the present invention. An external digital system **300** such as a host computer, personal computer and the like is coupled to the mass storage device **302** of the present invention. A logical block address is coupled via an address bus **306** to the volatile RAM array **100** and to a controller circuit **304**. Control signals are also coupled to the controller **304** via a control bus **308**. The volatile RAM array 1.00 is coupled for providing the physical block address to the nonvolatile RAM array **400**. The controller **304** is coupled to control both the volatile RAM **100**, the nonvolatile array **104**, and for the generation of all flags.

A simplified example, showing the operation of the write operation according to the present invention is shown in FIGS. **4** through **8**. Not all the information flags are shown to avoid obscuring these features of the invention in excessive detail. The data entries are shown using decimal numbers to further simplify the understanding of the invention. It will be apparent to one of ordinary skill in the art that in a preferred embodiment binary counting will be used.

FIG. **4** shows an eleven entry mass storage device according to the present invention. There is no valid nor usable data stored in the mass storage device of FIG. **4**. Accordingly, all the physical block addresses are empty. The data stored in the nonvolatile mass storage location '6' is filled and old. Additionally, location '9' is defective and cannot be used.

The host directs the mass storage device of the example to write data pursuant to the logical block address '3' and then to '4'. The mass storage device will first write the data associated with the logical block address '3'. The device determines which is the first unused location in the nonvolatile memory. In this example, the first empty location is location '0'.

Accordingly, FIG. **5** shows that for the logical block address '3', the corresponding physical block address '0' is stored and the used flag is set in physical block address '0'. The next empty location is location '1'. FIG. **6** shows that for the logical block address '4', the corresponding physical block address '1' is stored and the used flag is set in physical block address '1'.

The host instructs that something is to be written to logical block address '3' again. The next empty location is determined to be location '2'. FIG. **7** shows that the old flag in location '0' is set to indicate that this data is no longer usable, the used flag is set in location '2' and the physical block address in location '3' is changed to '2'.

Next, the host instructs that something is to be written to logical block address '4' again. The next empty location is determined to be location '3'. FIG. **8** shows that the old flag in location '1' is set to indicate that this data is no longer usable, the used flag is set in location '3' and the physical block address in location '4' is changed to '3'. (Recall that there is generally no relation between the physical block address and the data stored in the same location.).

FIG. **9** shows algorithm **1** according to the present invention. When the system of the present invention receives an instruction to program data into the mass storage (step **200**), then the system attempts to locate a free block (step **202**), i.e., a block having an unset (not programmed) used/free flag. If successful, the system sets the used/free flag for that block and programs the data into that block (step **206**).

If on the other hand, the system is unable to locate a block having an unset used/free flag, the system erases the flags (used/free and old/new) and data for all blocks having a set old/new flag and unset defect flag (step **204**) and then searches for a block having an unset used/free flag (step **202**). Such a block has just been formed by step **204**. The system then sets the used/flag for that block and programs the data file into that block (step **206**).

If the data is a modified version of a previously existing file, the system must prevent the superseded version from being accessed. The system determines whether the data file supersedes a previous data file (step **208**). If so, the system sets the old/new flag associated with the superseded block (step **210**). If on the other hand, the data file to be stored is a newly created data file, the step of setting the old/new flag (step **210**) is skipped because there is no superseded block. Lastly, the map for correlating the logical address **308**- to the physical addresses updated (step **212**).

By following the procedure outlined above, the overhead associated with an erase cycle is avoided for each write to the memory **104** except for periodically. This vastly improves the performance of the overall computer system employing the architecture of the present invention.

In the preferred embodiment of the present invention, the programming of the flash memory follows the procedure commonly understood by those of ordinary skill in the art. In other words, the program impulses are appropriately applied to the bits to be programmed and then compared to the data being programmed to ensure that proper programming has occurred. In the event that a bit fails to be erased or programmed properly, a defect flag **148** is set which prevent that block from being used again.

FIG. **10** depicts a digital system **500** such as a digital camera employing an alternative embodiment of the present invention. Digital system **500** is illustrated to include a host **502**, which may be a personal computer (PC) or simply a processor of any generic type commonly employed in digital systems, coupled to a controller circuit **506** for storing in and retrieving information from non-volatile memory unit **508**.

The controller circuit **506** may be a semiconductor (otherwise referred to as an "integrated circuit" or "chip") or optionally a combination of various electronic components. In the preferred embodiment, the controller circuit is depicted as a single chip device. The non-volatile memory unit **508** is comprised of one or more memory devices, which may each be flash or EEPROM types of memory. In the preferred embodiment of FIG. **10**, memory unit **508** includes a plurality of flash memory devices, **510-512**, each flash device includes individually addressable locations for storing information. In the preferred application of the embodiment in FIG. **10**, such information is organized in blocks with each block having one or more sectors of data. In addition to the data, the information being stored may further include status information regarding the data blocks, such as flag fields, address information and the like.

The host **502** is coupled through host information signals **504** to a controller circuit **506**.

The host information signals comprise of address and data busses and control signals for communicating command, data and other types of information to the controller circuit **506**, which in turn stores such information in memory unit **508** through flash address bus **512**, flash data bus **514**, flash signals **516** and flash status signals **518** (**508** and **513-516** collectively referred to as signals **538**). The signals **538** may provide command, data and status information between the controller **506** and the memory unit **508**.

The controller **506** is shown to include high-level functional blocks such as a host interface block **520**, a buffer RAM block **522**, a flash controller block **532**, a microprocessor block **524**, a microprocessor controller block **528**, a microprocessor storage block **530**, a microprocessor ROM block **534**, an ECC logic block **540** and a space manager block **544**. The host interface block **520** receives host information signals **504** for providing data and status information from buffer RAM block **522** and microprocessor block **524** to the host **502** through host information signals **504**. The host interface block **520** is coupled to the microprocessor block **524** through the microprocessor information signals **526**, which is comprised of an address bus, a data bus and control signals.

The microprocessor block **524** is shown coupled to a microprocessor controller block **528**, a microprocessor storage block **530** and a microprocessor ROM block **534**, and serves to direct operations of the various functional blocks shown in FIG. **10** within the controller **506** by executing program instructions stored in the microprocessor storage

block **530** and the microprocessor ROM block **534**. Microprocessor **524** may, at times, execute program instructions (or code) from microprocessor ROM block **534**, which is a non-volatile storage area. On the other hand, microprocessor storage block **530** may be either volatile, i.e., read-and-write memory (RAM), or non-volatile, i.e., EEPROM, type of memory storage. The instructions executed by the microprocessor block **524**, collectively referred to as program code, are stored in the storage block **530** at some time prior to the beginning of the operation of the system of the present invention. Initially, and prior to the execution of program code from the microprocessor storage location **530**, the program code may be stored in the memory unit **508** and later downloaded to the storage block **530** through the signals **538**. During this initialization, the microprocessor block **524** can execute instructions from the ROM block **534**.

Controller **506** further includes a flash controller block **532** coupled to the microprocessor block **524** through the microprocessor information signals **526** for providing and receiving information from and to the memory unit under the direction of the microprocessor. Information such as data may be provided from flash controller block **532** to the buffer RAM block **522** for storage (may be only temporary storage) therein through the microprocessor signals **526**.

Similarly, through the microprocessor signals **526**, data may be retrieved from the buffer RAM block **522** by the flash controller block **532**.

ECC logic block **540** is coupled to buffer RAM block **522** through signals **542** and further coupled to the microprocessor block **524** through microprocessor signals **526**. ECC logic block **540** includes circuitry for generally performing error coding and correction functions. It should be understood by those skilled in the art that various ECC apparatus and algorithms are commercially available and may be employed to perform the functions required of ECC logic block **540**. Briefly, these functions include appending code that is for all intensive purposes uniquely generated from a polynomial to the data being transmitted and when data is received, using the same polynomial to generate another code from the received data for detecting and potentially correcting a predetermined number of errors that may have corrupted the data. ECC logic block **540** performs error detection and/or correction operations on data stored in the memory unit **508** or data received from the host **502**.

The space manager block **544** employs a preferred apparatus and algorithm for finding the next unused (or free) storage block within one of the flash memory devices for storing a block of information, as will be further explained herein with reference to other figures. As earlier discussed, the address of a block within one of the flash memory devices is referred to as PBA, which is determined by the space manager by performing a translation on an LBA received from the host. A variety of apparatus and method may be employed for accomplishing this translation. An example of such a scheme is disclosed in U.S. Pat. No. 5,485,595, entitled "Flash Memory Mass Storage Architecture Incorporating Wear Leveling Technique Without Using CAM Cells", the specification of which is herein incorporated by reference. Other LBA to PBA translation methods and apparatus may be likewise employed without departing from the scope and spirit of the present invention.

Space manager block **544** includes SPM RAM block **548** and SPM control block **546**, the latter two blocks being coupled together. The SPM RAM block **548** stores the LBA-PBA mapping information (otherwise herein referred to as translation table, mapping table, mapping information, or table) under the control of SPM control block **546**. Alternatively, the SPM RAM block **548** may be located outside of the controller, such as shown in FIG. **3** with respect to RAM array **100**.

In operation, the host **502** writes and reads information from and to the memory unit **508** during for example, the performance of a read or write operation through the controller **506**. In so doing, the host **502** provides an LBA to the controller **506** through the host signals **504**. The LBA is received by the host interface block **520**. Under the direction of the microprocessor block **524**, the LBA is ultimately provided to the space manager block **544** for translation to a PBA and storage thereof, as will be discussed in further detail later.

Under the direction of the microprocessor block **524**, data and other information are written into or read from a storage area, identified by the PBA, within one of the flash memory devices **510-512** through the flash controller block **532**. The information stored within the flash memory devices may not be overwritten with new information without first being erased, as earlier discussed. On the other hand, erasure of a block of information (every time prior to being written), is a very time and power consuming measure. This is sometimes referred to as erase-before-write operation. The preferred embodiment avoids such an operation by continuously, yet efficiently, moving a sector (or multiple sectors) of information, within a block, that is being re-written from a PBA location within the flash memory to an unused PBA location within the memory unit **508** thereby avoiding frequent erasure operations. A block of information may be comprised of more than one sector such as 16 or 32 sectors. A block of information is further defined to be an individually-erasable unit of information. In the past, prior art systems have moved a block stored within flash memory devices that has been previously written into a free (or unused) location within the flash memory devices. Such systems however, moved an entire block even when only one sector of information within that block was being re-written. In other words, there is waste of both storage capacity within the flash memory as well as waste of time in moving an entire block's contents when less than the total number of sectors within the block are being re-written. The preferred embodiments of the present invention, as discussed herein, allow for "moves" of less than a block of information thereby decreasing the number of move operations of previously-written sectors, consequently, decreasing the number of erase operations.

Referring back to FIG. **10**, it is important to note that the SPM RAM block **548** maintains a table that may be modified each time a write operation occurs thereby maintaining the LBA-PBA mapping information and other information regarding each block being stored in memory unit **508**. Additionally, this mapping information provides the actual location of a sector (within a block) of information within the flash memory devices. As will be further apparent, at least a portion of the information in the mapping table stored in the SPM RAM block **548** is "shadowed" (or copied) to memory unit **508** in order to avoid loss of the mapping information when power to the system is interrupted or terminated. This is, in large part, due to the use of volatile memory for maintaining the mapping information. In this connection, when power to the system is restored, the portion of the mapping information stored in the memory unit **508** is transferred to the SPM RAM block **548**.

It should be noted, that the SPM RAM block **548** may alternatively be nonvolatile memory, such as in the form of flash or EEPROM memory architecture. In this case, the mapping table will be stored within nonvolatile memory thereby avoiding the need for "shadowing" because during

power interruptions, the mapping information stored in non-volatile memory will be clearly maintained.

When one or more sectors are being moved from one area of the flash memory to another area, the preferred embodiment of the present invention first moves the sector(s) from the location where they are stored in the flash memory devices, i.e., **510-512**, to the buffer RAM block **522** for temporary storage therein. The moved sector(s) are then moved from the buffer RAM block **522** to a free area within one of the flash memory devices. It is further useful to note that the ECC code generated by the ECC logic block **540**, as discussed above, is also stored within the flash memory devices **510-512** along with the data, as is other information, such as the LBA corresponding to the data and flag fields.

FIGS. **11-21** are presented to show examples of the state of a table **700** in SPM RAM block **548** configured to store LBA-PBA mapping information for identification and location of blocks (and sectors within the blocks) within the memory unit **508**. Table **700** in all of these figures is shown to include an array of columns and rows with the columns including virtual physical block address locations or VPBA block address locations **702**, move virtual physical address locations or MVPBA block address locations **704**, move flag locations **706**, used/free flag locations **708**, old/new flag locations **710**, defect flag locations **712** and sector move status locations **714**.

The rows of table include PBA/LBA rows **716**, **718** through **728** with each row having a row number that may be either an LBA or a PBA depending upon the information that is being addressed within the table **700**. For example, row **716** is shown as being assigned row number '00' and if PBA information in association with LBA '00' is being retrieved from table **700**, then LBA '00' may be addressed in SPM RAM block **548** at row **716** to obtain the associated PBA located in **730**. On the other hand, if status information, such as flag fields, **706-712**, regarding a block is being accessed, the row numbers of rows **716-728**, such as '00', '10', '20', '30', '40', '50', 'N−1' represent PBA, as opposed to LBA, values. Furthermore, each row of table **700** may be thought of as a block entry wherein each entry contains information regarding a block. Furthermore, each row of table **700** may be addressed by an LBA.

In the preferred embodiment, each block is shown to include 16 sectors. This is due to the capability of selectively erasing an entire block of 16 sectors (which is why the block size is sometimes referred to as an "erase block size". If an erase block size is 16 sectors, such as shown in FIGS. **11-21**, each block entry (or row) includes information regarding **16** sectors. Row **716** therefore includes information regarding a block addressed by LBA '00' through LBA '15' (or LBA '00' through LBA '0F' in Hex. notation). The next row, row **718**, includes information regarding blocks addressed by LBA '16' (or '10' in Hex.) through LBA '31' (or '1F' in Hex.) The same is true for PBAs of each block.

It should be noted however, other block sizes may be similarly employed. For example, a block may include 32 sectors and therefore have an erase block size of 32 sectors. In the latter situation, each block entry or row, such as **716**, **718**, **720** . . . , would include information regarding 32 sectors.

The VPBA block address locations **702** of table **700** stores information generally representing a PBA value corresponding to a particular LBA value. The MVPBA block address locations **704** store information representing a PBA value identifying, within the memory unit **508**, the location of where a block (or sector portions thereof) may have been moved. The move flag locations **706** store values indicating whether the block being accessed has any sectors that may have been moved to a location whose PBA is indicated by the value in the MVPBA block address location **704** (the PBA value within **704** being other than the value indicated in VPBA block address **702** wherein the remaining block address information may be located). The used/new flag location **708** stores information to indicate whether the block being accessed is a free block, that is, no data has been stored since the block was last erased. The old/new flag location **710** stores information representing the status of the block being accessed as to whether the block has been used and re-used and therefore, old. The defect flag location **712** stores information regarding whether the block is defective. If a block is declared defective, as indicated by the value in the defect flag location **712** being set, the defective block can no longer be used. Flags **708-712** are similar to the flags **110-114** shown and described with respect to FIG. **1**.

Sector move status location **714** is comprised of 16 bits (location **714** includes a bit for each sector within a block so for different-sized blocks, different number of bits within location **714** are required) with each bit representing the status of a sector within the block as to whether the sector has been moved to another block within the memory unit **508**. The moved block location within the memory unit **508** would be identified by a PBA that is other than the PBA value in VPBA block address location **702**. Said differently, the status of whether a sector within a block has been moved, as indicated by each of the bits within **714**, suggests which one of either the VPBA block address locations **702** or the MBPBA block address locations **704** maintain the most recent PBA location for that sector.

Referring still to FIG. **11**, an example of the status of the table **700** stored in SPM RAM block **548** (in FIG. **10**) is shown when, by way of example, LBA '0' is being written. As previously noted, in the figures presented herein, a block size of sixteen sectors (number 0-15 in decimal notation or 0-10 in hexadecimal notation) is used to illustrate examples only. Similarly, N blocks (therefore N LBAs) are employed, numbered from 0-N−1. The block size and the number of blocks are both design choices that may vary for different applications and may depend upon the memory capacity of each individual flash memory device (such as **510-512**) being employed. Furthermore, a preferred sector size of 512 bytes is used in these examples whereas other sector sizes may be employed without departing from the scope and spirit of the present invention.

Assuming that the operation of writing to LBA '0' is occurring after initialization or system power-up when all of the blocks within the flash memory devices **510-512** (in FIG. **10**) have been erased and are thus free. The space manager block **548** is likely to determine that the next free PBA location is '00'. Therefore, '00' is written to **730** in VPBA block address **702** of row **716** wherein information regarding LBA '0' is maintained, as indicated in table **700** by LBA row number '00'. Since no need exists for moving any of the sectors within the LBA 0 block, the MVPBA block address **704** for row **716**, which is shown as location **732** may include any value, such as an initialization value (in FIG. **11**, 'XX' is shown to indicate a "don't care" state).

The value in **734** is at logic state '0' to show that LBA '0' block does not contain any moved sectors. Location **736** within the used flag **708** column of row **716** will be set to logic state '1' indicating that the PBA '0' block is in use. The state of location **738**, representing the old flag **710** for row **716**, is set to '0' to indicate that PBA '0' block is not "old" yet. Location **740** maintains logic state '0' indicating that the PBA '0' block is not defective and all of the bits in move status

location **714** are at logic state '0' to indicate that none of the sectors within the LBA '0' through LBA '15' block have been moved.

In FIG. **11**, the status information for LBA '0' in row **716**, such as in move flag location **706**, used flag location **708**, old flag location **710**, defect flag location **712** and move status location **714** for all remaining rows, **716-728**, of table **700** are at logic state '0'. It is understood that upon power-up of the system and/or after erasure of any of the blocks, the entries for the erased blocks, which would be all blocks upon power-up, in table **700**, are all set to logic state '0'.

At this time, a discussion of the contents of one of the flash memory devices within the memory unit **508**, wherein the LBA '0' block may be located is presented for the purpose of a better understanding of the mapping information shown in table **700** of FIG. **11**.

Turning now to FIG. **22**, an example is illustrated of the contents of the flash memory device **510** in accordance with the state of table **700** (as shown in FIG. **11**). LBA '0', which within the memory unit **508** is identified at PBA '0' by controller **506** (of FIG. **10**) is the location wherein the host-identified block is written. A PBA0 row **750** is shown in FIG. **22** to include data in sector data location **752**. An ECC code is further stored in ECC location **754** of PBA0 row **750**. This ECC code is generated by the ECC logic block **540** in association with the data being written, as previously discussed. Flag field **756** in PBA0 row **750** contains the move, used, old and defect flag information corresponding to the sector data of the block being written. In this example, in flag field **756**, the "used" flag and no other flag is set, thus, flag field **756** maintains a logic state of '0100' indicating that PBA '0' is "used" but not "moved", "old" or "defective".

PBA0 row **750** additionally includes storage location for maintaining in LBA address location **758**, the LBA number corresponding to PBA '0', which in this example, is '0'. While not related to the example at hand, the remaining PBA locations of LBA '0' are stored in the next 15 rows following row **750** in the flash memory device **510**.

It will be understood from the discussion of the examples provided herein that the information within a PBA row of flash memory device **510** is enough to identify the data and status information relating thereto within the LBA '0' block including any moves associated therewith, particularly due to the presence of the "move" flag within each PBA row (**750**, **762**, **764**, . . .) of the flash memory. Nevertheless, alternatively, another field may be added to the first PBA row of each LBA location within the flash, replicating the status of the bits in the move status location **714** of the corresponding row in table **700**. This field is optionally stored in sector status location **760** shown in FIG. **22** to be included in the first PBA row of each LBA block, such as row **750**, **780** and so on. Although the information maintained in location **760** may be found by checking the status of the "move" flags within the flag fields **756** of each PBA row, an apparent advantage of using location **760** is that upon start-up (or power-on) of the system, the contents of table **700** in SPM RAM block **548** may be updated more rapidly due to fewer read operations (the reader is reminded that table **700** is maintained in SPM RAM **548**, which is volatile memory whose contents are lost when the system is power-down and needs to be updated upon power-up from non-volatile memory, i.e. memory unit **508**).

That is, rather than reading every PBA row (altogether 16 rows in the preferred example) to update each LBA entry of the table **700** upon power-up, only the first PBA row of each LBA must be read from flash memory and stored in SPM RAM **548** thereby saving time by avoiding needless read operations. On the other hand, clearly more memory capacity is utilized to maintain 16 bits of sector status information per LBA.

In the above example, wherein location **760** is used, the value in sector status location **760** would be all '0's (or '0000' in hexadecimal notation).

In flash memory device **510**, each of the rows **750**, **762**, **764**, **768** . . . , is a PBA location with each row having a PBA row number and for storing data and other information (data and other information are as discussed above with respect to row **750**) for a sector within a block addressed by a particular LBA. Furthermore, every sixteen sequential PBA rows represents one block of information. That is, PBA rows **750**, **762**, **764** through **768**, which are intended to show 16 PBA rows correspond to LBA 0 (shown as row **716** in table **700** of FIG. **11**) and each of the PBA rows maintains information regarding a sector within the block. The next block of information is for the block addressed by LBA '10' (in Hex.) whose mapping information is included in row **718** of table **700**, and which is stored in locations starting from '10' (in hexadecimal notation, or '16' in decimal notation) and ending at '1F' (in hexadecimal notation, or '31') in the flash memory device **510** and so on.

Continuing on with the above example, FIG. **12** shows an example of the state of table **700** when LBA 0 is again being written by the host. Since LBA 0 has already been written and is again being written without first being erased, another free location within the memory unit **508** (it may serve helpful to note here that the blocks, including their sectors, are organized sequentially and continuously through each of the flash memory devices of memory unit **508** according to their PBAs such that for example, the next flash memory device following device **510** picks up the PBA-addressed blocks where flash memory device **510** left off, an example of this is where flash memory device **510** includes PBAs of 0-FF (in Hex.) and the next flash memory device, which may be **512**, may then include 100-1FF (in Hex.)) is located by space manager **544** for storage of the new information. This free location is shown to be PBA '10' (in Hexadecimal notation, or 16 in decimal notation). In row **718**, where the entries for LBA '10' will remain the same as shown in FIG. **11** except the used flag in location **742** will be set (in the preferred embodiment, a flag is set when it is at logic state '1' although the opposite polarity may be used without deviating from the present invention) to indicate that the PBA '10' is now "in use".

The entries in row **716** are modified to show '10' in MVPBA block address location **732**, which provides the PBA address of the moved portion for the LBA '00' block. The move flag in location **734** is set to logic state '1' to indicate that at least a portion (one or more sectors) of the LBA '00' block have been moved to a PBA location other than the PBA location indicated in location **730** of table **700**. Finally, the bits of the move status location **714** in row **716** are set to '1000000000000000' (in binary notation, or '8000' in hexadecimal notation), reflecting the status of the moved sectors within the block LBA '00'. That is, in this example, '8000' indicates that the first sector, or sector '0', within LBA '00' block has been moved to a different PBA location.

Referring now to FIG. **22**, the state of table **700** in FIG. **12** will affect the contents of the flash memory device **510** in that the moved sector of the LBA '0' block will now be written to PBA '10' in row **780**. Row **780** will then include the data for the moved sector, which is 512 bytes in size. With respect to the moved sector information, row **780** further includes ECC code, a copy of the values in flag locations **734-740** of table **700** (in FIG. **12**), and LBA '00' for indicating that the data in

row **780** belongs to LBA '00' and may further include the move status for each of the individual sectors within the LBA '0' block.

While not specifically shown in the figure, the move flag within location **756** of PBA row **750** is set to indicate that at least a portion of the corresponding block has been moved. The value stored in the move status location **714** of row **716** (in FIG. **12**), which is '8000' in Hex., is also stored within location **760** of the row **750**. As earlier noted, this indicates that only sector '0' of PBA '0' was marked "moved" and the new block LBA '0' was written to PBA '10' in flash memory. Without further detailed discussions of FIG. **22**, it should be appreciated that the examples to follow likewise affect the contents of the flash memory device **510**.

FIG. **13** shows the status of table **700** when yet another write operation to LBA '00' is performed. The values (or entries) in row **716** remain the same as in FIG. **12** except that the value in location **732** is changed to '20' (in Hex. Notation) to indicate that the moved portion of block LBA '00' is now located in PBA location '20' (rather than '10' in FIG. **12**). As in FIG. **12**, the value in move status location **714**, '8000', indicates that the first sector (with PBA '00') is the portion of the block that has been moved.

Row **718** is modified to show that the LBA '10' block is now old and can no longer be used before it is erased. This is indicated by the value in location **744** being set to logic state '1'. The entries for LBA '20', row **720**, remain unchanged except that location **746** is modified to be set to logic state '1' for reflecting the state of the PBA '20' block as being in use. It is understood that as in FIGS. **11** and **12**, all remaining values in table **700** of FIG. **13** that have not been discussed above and are not shown as having a particular logic state in FIG. **13** are all unchanged (the flags are all set to logic state '0').

Continuing further with the above example, FIG. **14** shows the state of table **700** when yet another write to LBA '0' occurs. For ease of comparison, there is a circle drawn around the values shown in FIG. **14**, which are at a different logic state with respect to their states shown in FIG. **13**. In row **716**, everything remains the same except for the new moved location, indicated as PBA '30', shown in location **732**. PBA '30' was the next free location found by the space manager **544**. As previously noted, this value indicates that a portion of the block of LBA '0' is now in PBA '30'; namely, the first sector (shown by the value in **714** of row **716** being '8000') in that block has been moved to PBA '30' in the flash memory device **510**.

Row **718** remains the same until it is erased. The flags in locations **742** and **744** are set to logic state '0'. Row **720** also remains unchanged except for the value in its old flag **710** column being modified to '1' to show that the block of PBA '20' is also old and can not be used until first erased. Row **722** remains the same except for the value in its used flag **708** column being changed to logic state '1' to show that the block of LBA '30' is now in use.

FIG. **15** is another example of the state of table **700**, showing the state of table **700** assuming that the table was at the state shown in FIG. **13** and followed by the host writing to LBA '5'. Again, the changes to the values in table **700** from FIG. **13** to FIG. **15** are shown by a circle drawn around the value that has changed, which is only one change.

When writing to LBA '5', it should be understood that the LBA entries of rows **716**, **718**, **720**, etc. are only for LBA '00', LBA '10', LBA '20', so on, and therefore do not reflect an LBA '5' entry. The reader is reminded that each of the LBA row entries is for a block of information with each block being 16 sectors in the preferred embodiment. For this reason, LBA '5' actually addresses the fifth sector in row **716**. Since PBA '20' was used to store LBA '0', only the sector within PBA '20', corresponding to LBA '5', is yet not written and "free". Therefore, the data for LBA '5' is stored in PBA '20' in sector '5'. The move status location **714** of row **716** will be modified to logic state '8400' (in Hex. Notation). This reflects that the location of the first and fifth sectors within LBA '0' are both identified at PBA '20' in the flash memory device **510**. The remaining values in table **700** of FIG. **15** remain the same as those shown in FIG. **13**.

FIGS. **16-18** show yet another example of what the state of table **700** may be after either power-up or erasure of the blocks with the memory unit **508**. In FIGS. **16** and **17**, the same write operations as those discussed with reference to FIGS. **11** and **12** are performed. The state of table **700** in FIGS. **16** and **17** resembles that of FIGS. **11** and **12**, respectively (the latter two figures have been re-drawn as FIGS. **16** and **17** for the sole convenience of the reader). Briefly, FIG. **16** shows the state of table **700** after a write to LBA '0' and FIG. **17** shows the state of table **700** after another write to LBA '0'.

FIG. **18** picks up after FIG. **17** and shows the state of table **700** after the host writes to LBA '5'. As indicated in FIG. **18**, LBA '5' has been moved to PBA '10' where LBA '0' is also located. To this end, MBPBA block address location **732** is set to '10' in row **716** and the move flag is set at location. **734** in the same row. Moreover, the state of move status location **714** in row **716** is set to '8400' (in Hex.) indicating that LBA '0' and LBA '5' have been moved, or that the first and fifth sectors within LBA '00' are moved. Being that these two sectors are now located in the PBA '10' location of the flash memory device **510**, the move flag for each of the these sectors are also set in the flash memory device **510**. It should be understood that LBA '5' was moved to PBA '10' because remaining free sectors were available in that block. Namely, even with LBA '0' of that block having been used, 15 other sectors of the same block were available, from which the fifth sector is now in use after the write to LBA '5'.

Continuing on with the example of FIG. **18**, in FIG. **19**, the state of the table **700** is shown after the host writes yet another time to LBA '0'. According to the table, yet another free PBA location, '20', is found where both the LBA '5' and LBA '0' are moved. First, LBA '5' is moved to the location PBA '10' to PBA '20' and then the new block of location LBA '0' is written to PBA '20'. As earlier discussed, any time there is a move of a block (for example, here the block of LBA '5' is moved) it is first moved from the location within flash memory where it currently resides to a temporary location within the controller **506**, namely with the buffer RAM block **522**, and then it is transferred from there to the new location within the flash memory devices.

The used flag in location **746** of row **720** is set to reflect the use of the PBA '20' location in flash memory and the old flag in location **744** is set to discard use of PBA '10' location until it is erased. Again, in flash memory, the state of these flags as well as the state of the move flag for both the LBA '0' and LBA '5' sectors are replicated.

FIG. **20** picks up from the state of the table **700** shown in FIG. **18** and shows yet another state of what the table **700** may be after the host writes to LBA '5'. In this case, the block of LBA '0' is first moved from location PBA '10' within the flash memory device **510** wherein it is currently stored to location PBA '20' of the flash memory. Thereafter, the new block being written to LBA '5' by the host is written into location PBA '20' of the flash memory. The flags in both table **700** and corresponding locations of the flash memory device **510** are accordingly set to reflect these updated locations.

FIG. **21** also picks up from the state of the table **700** shown in FIG. **18** and shows the state of what the table **700** may be after the host writes to LBA '7'. In this case, the new block is simply written to location PBA '10' of the flash memory since that location has not yet been used. Additionally, three of the bits of the move status location **714** in row **716** are set to show that LBA '0', LBA '5' and LBA '7' have been moved to another PBA location within the flash memory. Location **732** shows that the location in which these three blocks are stored is PBA '10'.

As may be understood from the discussion presented thus far, at some point in time, the number of sectors being moved within a block makes for an inefficient operation. Thus, the need arises for the user to set a threshold for the number of sectors within a block that may be moved before the block is declared "old" (the old flag is set) and the block is no longer used, until it is erased. This threshold may be set at, for example, half of the number of sectors within a block. This is demonstrated as follows: For a block having 16 sectors, when 8 of the sectors are moved into another block, the "original" block and the "moved" block (the block in which the moved sectors reside) are combined into the same PBA block. The combined PBA block may be stored in a new block altogether or, alternatively, the "original" block may be combined with and moved into the "moved" block. In the latter case, the "original" block is then marked as "old" for erasure thereof. If the combined PBA block is stored in a new block, both of the "original" and the "moved" blocks are marked as "old".

FIG. **23** depicts a general flow chart outlining some of the steps performed during a write operation. It is intended to show the sequence of some of the events that take place during such an operation and is not at all an inclusive presentation of the method or apparatus used in the preferred embodiment of the present invention.

The steps as outlined in FIG. **23** are performed under the direction of the microprocessor block **524** as it executes program code (or firmware) during the operation of the system.

When the host writes to a block of LBA M, step **800**, the space manager block **544**, in step **802**, checks as to whether LBA M is in use by checking the state of the corresponding used flag in table **700** of the SPM RAM block **548**. If not in use, in step **804**, a search is performed for the next free PBA block in memory unit **508**. If no free blocks are located, an "error" state is detected in **808**. But where a free PBA is located, in step **806**, its used flag is marked (or set) in table **700** as well as in flash memory. In step **810**, the PBA of the free block is written into the VPBA block address **702** location of the corresponding LBA row in table **700**.

Going back to step **802**, if the LBA M block is in use, search for the next free PBA block is still conducted in step **812** and upon the finding of no such free block, at **814**, an "error" condition is declared. Whereas, if a free PBA location is found, that PBA is marked as used in table **700** and flash memory, at step **816**. Next, in step **818**, the state of the block is indicated as having been moved by setting the move flag as well as the setting the appropriate bit in the move status location **714** of table **700**. The new location of where the block is moved is also indicated in table **700** in accordance with the discussion above.

Finally, after steps **818** and **810**, data and all corresponding status information, ECC code and LBA are written into the PBA location within the flash memory.

As earlier indicated, when a substantial portion of a block has sectors that have been moved (in the preferred embodiment, this is eight of the sixteen sectors), the block is declared "old" by setting its corresponding "old" flag. Periodically, blocks with their "old" flags set, are erased and may then be re-used (or re-programmed, or re-written).

As can be appreciated, an advantage of the embodiments of FIGS. **10-23** is that a block need not be erased each time after it is accessed by the host because if for example, any portions (or sectors) of the block are being re-written, rather than erasing the block in the flash memory devices or moving the entire block to a free area within the flash, only the portions that are being re-written need be transferred elsewhere in flash, i.e. free location identified by MVPA block address. In this connection, an erase cycle, which is time consuming is avoided until later and time is not wasted in reading an entire block and transferring the same.

Although the present invention has been described in terms of specific embodiments, it is anticipated that alterations and modifications thereof will no doubt become apparent to those skilled in the art. It is therefore intended that the following claims be interpreted as covering all such alterations and modification as fall within the true spirit and scope of the invention.

What is claimed is:

1. A method of updating sectors of data in a nonvolatile storage system comprising:

receiving a command from a host for updating one or more sectors of data into a location within the nonvolatile memory storage, said location defined by a particular block having previously-written one or more sectors of data, said particular block caused to be identified by a first group of logical block addresses, corresponding to a group of sectors, and including two or more sectors of data, each sector for storing a sector of data;

selecting one or more new blocks within the nonvolatile storage;

writing said updated one or more sectors of data to one or more new blocks caused to be identified by said first group of logical block addresses without copying the previously-written sectors of data in the particular block that are not updated; and

in response to receiving a new write command from the host to rewrite a second sector of data defined to be a sector of data within the particular block that is other than the first sector of data, rewriting the second sector of data into the one or more new blocks identified by the first group of logical block addresses without moving and copying the previously written sectors of data in the particular block that are not updated.

2. The method of claim **1**, further comprising receiving additional commands from the host for further updating, one or more times, sectors of data, wherein the previously-written sectors of data in the particular block are not moved every time a sector of data is updated.

3. The method of claim **1**, further comprising moving the previously-written sectors of data that are not updated from the particular block after writing the updated one or more sectors of data to one or more new blocks.

4. The method of claim **3**, further comprising erasing the particular block after moving the previously-written sectors of data that are not updated from the particular block.

5. The method of claim **1**, further comprising writing said updated one or more sectors of data to one or more new blocks caused to be identified by said first group of logical block addresses without moving the previously-written sectors of data in the particular block that are not updated.

6. The method of claim **1**, wherein receiving the command from the host further comprises receiving the first group of logical block addresses from the host.

7. A method of updating sectors of data in a nonvolatile storage system comprising:

receiving a command from a host for updating one or more sectors of data into a location within the nonvolatile memory storage, said location defined by a particular block having previously-written one or more sectors of data, said particular block caused to be identified by a first group of logical block addresses, corresponding to a group of sectors, and including two or more sectors of data, each sector for storing a sector of data;

selecting one or more new blocks within the nonvolatile storage;

writing said updated one or more sectors of data to one or more new blocks caused to be identified by said first group of logical block addresses without copying the previously-written sectors of data in the particular block that are not updated;

receiving additional commands from the host for further updating, one or more times, sectors of data;

moving the previously-written sectors of data that are not updated from the particular block after writing the updated one or more sectors of data to one or more new blocks; and

in response to receiving a new write command from the host to rewrite a second sector of data defined to be a sector of data within the particular block that is other than the first sector of data, rewriting the second sector of data into the one or more new blocks identified by the first group of logical block addresses without moving and copying the previously written sectors of data in the particular block that are not updated.

8. The method of claim 7, further comprising erasing the particular block after moving the previously-written sectors of data that are not updated from the particular block.

9. The method of claim 7, wherein receiving the command from the host further comprises receiving the first group of logical block addresses from the host.

10. A method of updating sectors of data in a nonvolatile storage system comprising:

receiving a command and a first group of logical block addresses from a host for updating one or more sectors of data into a location within the nonvolatile memory storage, said location defined by a particular block having previously-written one or more sectors of data, said particular block caused to be identified by the first group of logical block addresses, corresponding to a group of sectors, and including two or more sectors of data, each sector for storing a sector of data;

selecting one or more new blocks within the nonvolatile storage;

writing said updated one or more sectors of data to one or more new blocks caused to be identified by said first group of logical block addresses without moving and without copying the previously-written sectors of data in the particular block that are not updated;

moving the previously-written sectors of data that are not updated from the particular block after writing the updated one or more sectors of data to one or more new blocks; and

in response to receiving a new write command from the host to rewrite a second sector of data defined to be a sector of data within the particular block that is other than the first sector of data, rewriting the second sector of data into the one or more new blocks identified by the first group of logical block addresses without moving and copying the previously written sectors of data in the particular block that are not updated.

11. The method of claim 10, further comprising erasing the particular block after moving the previously-written sectors of data that are not updated from the particular block.

* * * * *